

POLITECNICO DI MILANO  
Dipartimento di Elettronica, Informazione e Bioingegneria  
Master of Science in Computer Science Engineering



MS Project Sharepoint Add-in  
MS Project and Wunderlist Integration

**Supervisor:** Prof. Cinzia CAPPIELLO

**Master Graduation Thesis of:**  
Mohamed ABOELKHEIR  
Matricola N. 820829

A.A. 2015/2016

---

## Acknowledgements

I would like to thank, before all, my supervisor Prof. Cinzia Cappiello for all her understanding, cooperation, availability, and guidance during the development of this work.

I would like to give special thanks the company Allinace s.r.l, which gave me the opportunity of doing a stage at their office, where I developed this work. In specific, my adviser: Matteo Marino, R&D director: Fabrizio Brusadelli for his availability and help understand MS Project software architecture, fellow programmer: Tibor Bacsı for technical help.

I thank GREEN IT Erasmus program for granting me a scholarship for studying at the Politecnico di Milano.

I would like to direct great thanks and gratitude to my family for all the support they have given me before and after coming to Italy, which was vital for begin able to complete it.

Moreover, I heartily thank all the friends: Italians, Egyptians, and international, within and from outside of Politecnico di Milano. They have provided me great company, which has been vital for sustaining a satisfying social life which has also driven me for a better academic performance.

Milan, September 2016

---

## Abstract

Project Portfolio Management has become a key success factor and a necessity for managing many projects in companies at the same time. Therefore, there a lot of successful and well performing tools that allow companies to perform this task efficiently. Since, it is a huge market with a lot of demand, many big software companies such as Microsoft, Project Objects, and CA Technologies already targeted this market and developed their own unique solutions to fulfill this market need. Each tool has its own structure in tackling the issue, but at the end, they seek to provide similar functionalities manifested in managing data, plans, tasks, resources, and deadlines of the different projects. However, the fundamental theories of PPM remain the same regardless of different implementations. One of the key factors of increasing the efficiency and effectiveness of all these projects and the process of PPM in general is having real-time data entered into the system. For this reason to take place, the tool needs to provide high level of usability and accessibility to employees during entering data. PPM tools are quite complex and provide many functionalities and that is why they are mainly accessed by browsers or desktop based applications. In this sense, these tools can benefit from usable apps for hand-held devices, especially for users who enter the bulk of data to the system. This would provide more up-to-date data to the system which serves the real-time requirement they seek satisfying. Instead of developing a specific app for such tools, one could, however, integrate the tool with an already existing task management tool. Such task management tools are usually much simpler, user-friendly, focused on managing tasks only and many have already usable hand-held device apps built for them. The target of this work is to enrich MS Project by integrating it with Wunderlist, a task management tool owned by Microsoft.

# Contents

<b>Acknowledgements</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>List of Figures</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 State of Art</b>	<b>3</b>
2.1 PPM Tools . . . . .	3
2.2 Comparison of the tools . . . . .	5
2.3 Wunderlist . . . . .	6
2.4 Conclusions . . . . .	7
<b>3 Solution Theory</b>	<b>8</b>
3.1 Product Functions . . . . .	8
3.2 Approach . . . . .	9
3.2.1 Concept . . . . .	9
3.2.2 User interface . . . . .	10
3.3 Functional Requirements . . . . .	10
3.3.1 Scenarios: . . . . .	10
3.3.2 Flow Chart Models . . . . .	13
<b>4 Implementation</b>	<b>17</b>
4.1 Libraries used . . . . .	17
4.2 Add-in Development . . . . .	17
4.2.1 Add-in development Tool . . . . .	17
4.2.2 Add-in added items . . . . .	19
4.2.3 User interface . . . . .	20
4.3 Communication with Wunderlist . . . . .	25
4.3.1 Registration of the Sharepoint Add-in . . . . .	25
4.3.2 Authenticating the Sharepoint Add-in and obtaining credentials	27
4.3.3 Making calls to Wunderlist . . . . .	28
4.3.4 GET calls: . . . . .	28
4.3.5 POST calls: . . . . .	28
4.3.6 PATCH calls: . . . . .	29
4.4 Communication with Project online/server . . . . .	29
4.4.1 Retrieve project information: . . . . .	30
4.4.2 GET Calls . . . . .	30
4.4.3 POST Calls: . . . . .	30
4.5 Non-functional requirement implementations . . . . .	31

4.5.1	Check license . . . . .	31
4.5.2	Minifying and obfuscating the JavaScript files . . . . .	31
4.6	Testing . . . . .	31
4.6.1	Test Cases for the first button . . . . .	32
4.6.2	Test Cases for the second button . . . . .	34
4.6.3	Test Cases for all three buttons . . . . .	34
4.6.4	Test Cases for the fourth "Guide" Button . . . . .	35
4.7	Limitations, Difficulties, and Work-arounds: . . . . .	36
<b>5</b>	<b>Conclusions and future work</b>	<b>38</b>
5.1	Conclusions . . . . .	38
5.2	Future work . . . . .	39
<b>6</b>	<b>Appendix</b>	<b>40</b>
6.1	Coummunication with Wunderlist . . . . .	40
6.1.1	GET calls . . . . .	40
6.1.2	POST calls . . . . .	41
6.1.3	PATCH calls . . . . .	42
6.2	Communication with Project online/server . . . . .	43
6.2.1	Retrieve project information: . . . . .	43
6.2.2	GET Calls . . . . .	43
6.2.3	POST Calls: . . . . .	44
6.3	Non-functional requirements implementations . . . . .	46
6.3.1	Check license . . . . .	46
6.3.2	Minifying and obfuscating the JavaScript files . . . . .	48
	<b>References</b>	<b>49</b>

## List of Figures

1	MS Project . . . . .	3
2	CA PPM . . . . .	5
3	Wunderlist . . . . .	6
4	Use Case Chart . . . . .	8
5	Authentication Flowchart . . . . .	13
6	The first button's flowchart . . . . .	14
7	The second button's flowchart . . . . .	15
8	The third button's flowchart . . . . .	16
9	Add-in Code Structure . . . . .	18
10	External App registration in Wunderlist . . . . .	19
11	Process of Add-in Registration in Wunderlist . . . . .	20
12	Project page of MS Project . . . . .	21
13	MS Project modal . . . . .	22
14	Modal shown for the First Button . . . . .	23
15	Modal shown for the Second Button . . . . .	23
16	Modal shown for the Third Button . . . . .	24
17	The project's corresponding list in Wunderlist . . . . .	25
18	Add-in Registration . . . . .	26
19	Add-in authorization Page at Wunderlist . . . . .	27

## 1 Introduction

In the current settings of business, companies have to keep innovating and growing; otherwise, they lose their competitiveness and fail. Therefore, companies' upper management needs to put effective strategic business objectives and decide the directions of growth of their companies. One of the most successful approaches that facilitate working to meet these strategic objectives for managers is Project portfolio management. Normally, a company's portfolio includes the projects, programs, and other works the company is undertaking for a specific business objective. Hence, Project Portfolio Management is managing all these projects in the same portfolio and making decisions about their priority, timing, cost, scope, resources, etc. It is, also, concerned with monitoring the progress of the projects involved, performing reporting, measuring KPIs and providing the key information to the responsible people, and communicating changes and notifications to the concerned people in due time. Since this is a pretty complicated process already, many big firms concentrated on this market and provided quite complex solutions to automatize these tasks. Some of the biggest players in this market are Project Objects, CA PPM, and MS Project by Microsoft. In addition, automatization and speed are quite important in this process because the more up-to-date the data in the information system is, the faster information is analyzed, key decisions are taken, and urgent issues solved. Due to the fact that businesses are quite different by nature, there is no magical one-size fits all software. Instead, many of the available software are highly customizable and they usually get personalized and customized by third-party consultancy companies to represent the business models of the companies deploying them. This appears quite evidently while examining the software examples previously mentioned; for example, CA PPM allows extensive customization to its structure, database, and through workflows "processes", which can run executable scripts and export external libraries. Microsoft provides a store for MS Project where Apps could be added to customize and provide new functionalities to the base product they have. On the other hand, the fact that these programs are directed towards high-level management make them mainly web-based or cloud-based software accessible by browsers and no suitable Mobile App versions are available. The reason is usually due to the complexity of these projects and the fact that they manage a lot of different types of information of the businesses. At the downside, though, this means that employees working on specific tasks would need to wait until they log-in the system using a computer to enter the tasks' progress data to the system. For such situations, there is no denial that a mobile friendly version would be highly desirable due to the fact that the quality of the information in the system depends on how immediate, up-to-date, and real-time the Information System's data is. Meanwhile, some other available tools are more focused on personal task management. Such tools usually offer high level of usability by providing mobile Apps. They, usually, also allow the insertion of more details and attributes for tasks to allow better task management.

For instance, Wunderlist is a task management tool that is owned by Microsoft; it provides a convenient Mobile App and a broad range of functionalities for managing tasks. This means that the data on Wunderlist are always up-to-date because it could be used by employees as the main tool for registering their progress. From here emerges the value of the integration between a PPM solution and a task management solution that would provide both the usability for technical employees and workers, and the immediate insertion of up-to-date data to the PPM software. Indeed, this is the scope of this work the integration of a PPM tool and a task management tool, both owned by Microsoft, respectively MS Project and Wunderlist.

This document is structured in four more sections. Section 2 presents some of the popular PPM tools and sheds more light on the problem and the gap these tools have compared to task management tools. Section 3 presents the solution developed in this thesis with the functional requirement analysis and the flow charts of the actions in the solution. Section 4 describes the implementation phase including all the main developed components of the software, test cases, difficulties faced, and proposed future features to the solution proposed. Then, the conclusion section gives a sum-up of the whole work and discusses its results. Finally, some of the code snippets of the main components are presented in the appendix to further elaborate the implementation part.



## 2 State of Art

This section starts by giving an idea about the general nature of the PPM tools by presenting some of the available and popular ones, specifically, MS Project and CA PPM. Then, it proceeds to explain the problems these tools have and discussing a proper solution to solve them for the case MS Project.

### 2.1 PPM Tools

A lot of very comprehensive project portfolio management and project management tools are available on the market. One of the most successful tools is MS Project, a project management tool developed by Microsoft. MS Project could be used in its cloud-based form or a company could choose to deploy an on-premise version. The cloud-based, called "Microsoft Project Online", is typically automatically up-to-date with the latest software updates by Microsoft. However, the on-premise is released in versions installable to the clients' servers, called "Microsoft Project Server"; the latest of which are Project Server 2016 and 2013 beforehand.

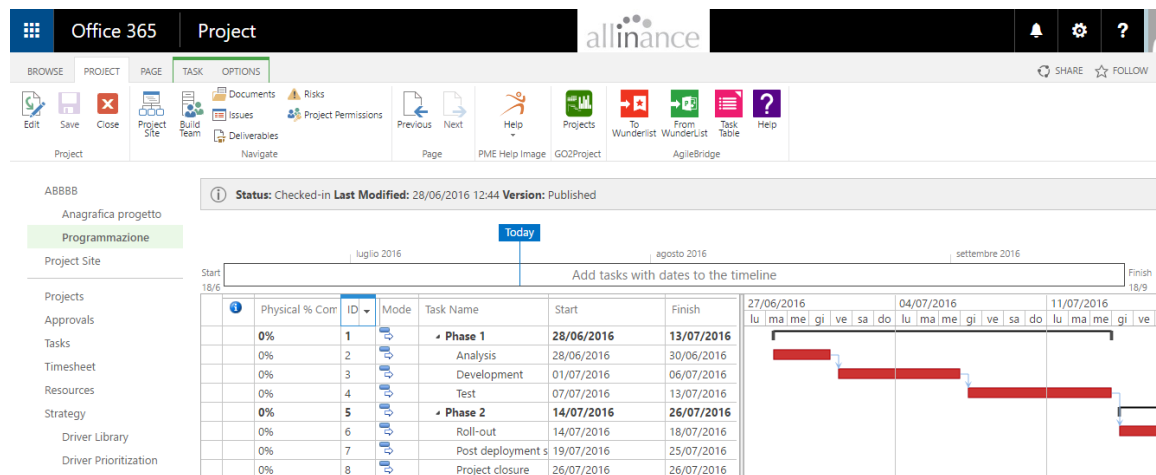


Figure 1: MS Project

MS Project<sup>1</sup> provides a wide range of functionalities for managing projects and resources. As part of managing resources, as seen in Figure 1, the user can view the tasks assigned to him/her by clicking on the tasks tab. This would display all the tasks and their details - including start, finish, completion, etc. - from all the

<sup>1</sup>Tool's Website <https://products.office.com/en/project/enterprise-project-server>

different projects that involve this user. Clicking on the timesheet tab would, however, display all the hours the user entered for each of his/her assigned tasks beside the essential details of those tasks. The functionalities provided by MS Project as part managing projects are also evident in Figure 1, which demonstrates a specific project's page in MS Project. Namely, it shows the list of tasks the project consists of besides showing for each task its start, end, physical completion, cost, and the resources it is assigned to. The page also provides a visualization of the schedule of the project according to the timeline of its tasks. At the same time, the tool manages the financial aspect of the projects as the information registry page of the project contains its metadata plus its owner, status, total budget, baseline cost, and actual cost. Among other technical functionalities, the tool manages user permissions, exporting and importing data and projects from or to Excel, etc. The interesting feature in this tool is that it uses Microsoft SharePoint as its foundation, and the page seen in Figure 1 could be viewed through the Project Web App (PWA) component built on the Sharepoint. This is fundamental in our work because it allows the addition of Microsoft Sharepoint Add-ins installed on the Sharepoint to the Project Web App, which in this case could even provide more functionalities beside the default ones provided in MS Project.

Another popular tool is CA PPM made by CA Technologies. CA PPM, in principle, provides most of the functionalities MS Project provides, in addition to more complex features. For example, CA PPM installations would include a big database that saves the data about users and projects. Moreover, CA PPM interface is very customizable using portlets as seen in Figure 2. These portlets are customized views that execute queries in the database to view specific information required by the user. Figure 2 shows some of the many functionalities this tool provide to its users in order to manage projects and resources. The tool can also implement business process and workflows and grouping projects in programs or portfolios, while managing their finances, risks, issues. It is also able to display customizable dashboards and reports of the data in the system. It could be integrated with other tools such as SAP so as to import or export data[1].

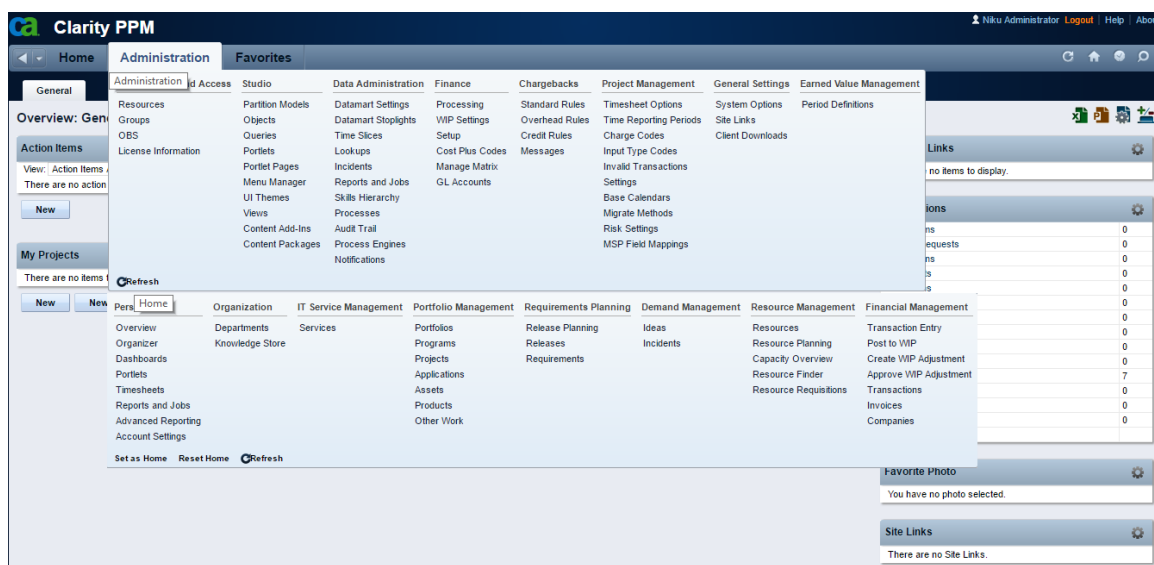


Figure 2: CA PPM

## 2.2 Comparison of the tools

Examining those two tools above, namely MS Project and CA PPM, it is quite noticeable how PPM tools, in general, are quite complex providing many diverse tasks to their users. This also stems from the fact that they are designed specifically to assist project managers to develop plans, keep track of the company's progress, as well as analyze cost and times for future planning. And this complexity forces the interaction with these tools to be usually done through a browser or desktop installable software. This leaves behind a lot to be required in terms of usability. It is more convenient if these tools would provide several apps for the most crucial or the most used functionalities. In this way the users do not need to wait until they access the tool from a browser or from a personal computer, but they can access when they want from their hand-held devices. This is even more needed for the functionalities related to employees of the company, from whom the bulk of the tool's data come from, rather than the project manager himself/herself. While project managers and upper management are the target users of the tools, they are not the only users of these tools because the tool, at the end, analyzes data of, and possibly entered by, all employees of the companies. Therefore, large percentage of the users will only be using simple functionalities; namely, those of managing the tasks of the projects they are working on and their timesheets. Consequently, it would make more sense if for such simple highly usable functionalities a more usable interface is provided so as to make it simpler and easier for those users to enter data to the tool. This also implies that if employees have such option, they would more probably enter the data

once they have them, which will make the system real-time. Being real-time is one of the key points of making a successful Information System for a company.

In fact MS Project provides an app for timesheet, which employees can install and can fill their timesheet for their assigned tasks. CA PPM provides two apps: one for timesheets and another for viewing personal reports. But, both tools are yet to provide any apps for managing tasks and task completion.

In summary, the issue concerns the fact that the Project management and PPM software tools are mainly presented as desktop applications and they do not have mobile apps through which the user can interact with them. On the other hand, a different typology of tools exists that specializes in managing personal tasks. An example of such tools is described in the next section.

## 2.3 Wunderlist

Wunderlist is a cloud-based task management tool that has been already acquired by Microsoft. As seen in Figure 3, the app manages personal lists of tasks. One can share lists of tasks with other users. Task could be assigned deadlines, have a person that it is assigned to, include notes, or have files attached to them. They could even be further subdivided into subtasks for partial completion. The tool has a very usable native app which user interface is very similar to the desktop one, easy, simple to use, and convenient. Quite importantly, since it is acquired by Microsoft, the App is also provided in Microsoft sharepoint.

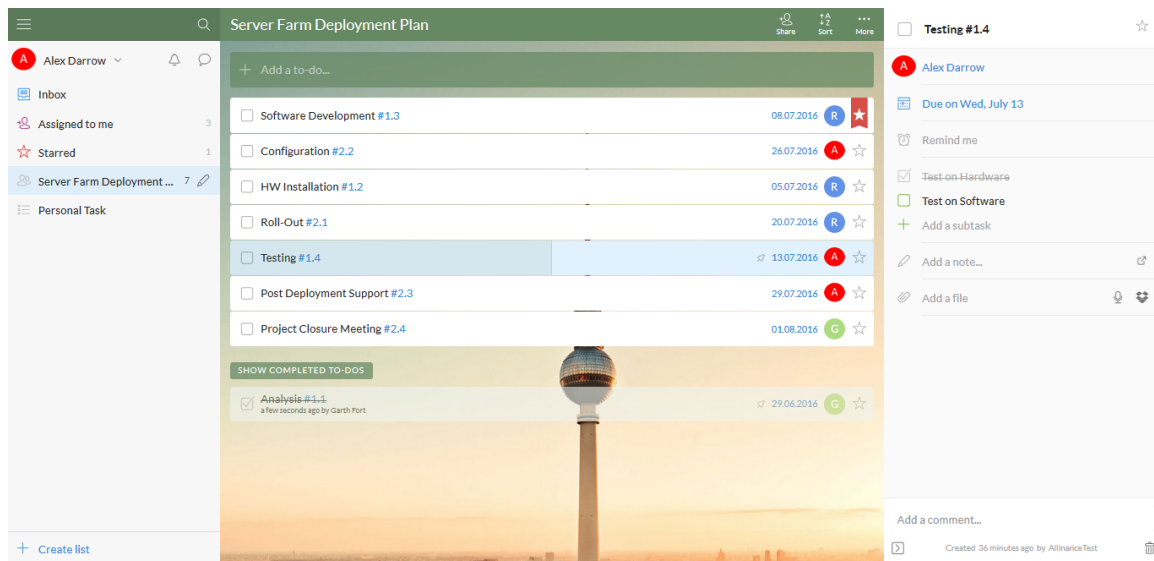


Figure 3: Wunderlist

## 2.4 Conclusions

After having examined both types of tools, one can now see the gap between PPM tools and task management solutions. From one side, task management solutions strongly address and efficiently manage personal tasks providing a usable, user-friendly app that facilitates fast marking and completion of tasks when required. However, updating project task completion or their details on PPM tools is not feasible using mobile devices and requires accessing the tool through a browser or software on the user's personal computer.

One general solution to tackle the gap for PPM tools is developing task management Apps for of these tools, but this is risky in itself as such an app will have to compete with the company's native app, when they decide to develop one. A more viable solution for the case of MS Project emerges from leveraging the already implemented mobile interfaces provided by Wunderlist to serve the project management software such as MS Project. This could be done by integrating both tools together. At the same time, MS Project offers a high degree of customization through the office store where Sharepoint add-ins could be published. Then, MS project customers could install any of the add-ins to their version of the MS Project through their "site contents" tab. At that point, the Add-in to be built will communicate with Wunderlist using its readily available APIs. The fact that Wunderlist is also part of Microsoft Sharepoint gives the Add-in a strong economic advantage as well. The development of such Add-in is what was undertaken by this work and is explained in the next sections, namely, the Solution theory and Implementation sections.

### 3 Solution Theory

In this chapter, a detailed explanation is given on the approach, decisions, and steps used to develop the solution proposed in the previous section.

#### 3.1 Product Functions

**Use Case Model** The following diagram shows the use cases for the add-in. In our case, the add-in is supposed to be used by the resources of MS Project. Their capabilities are performing any of the three main activities provided by clicking on one of the main three buttons. As well, they can click the fourth "guide" button to download the Add-in guide.

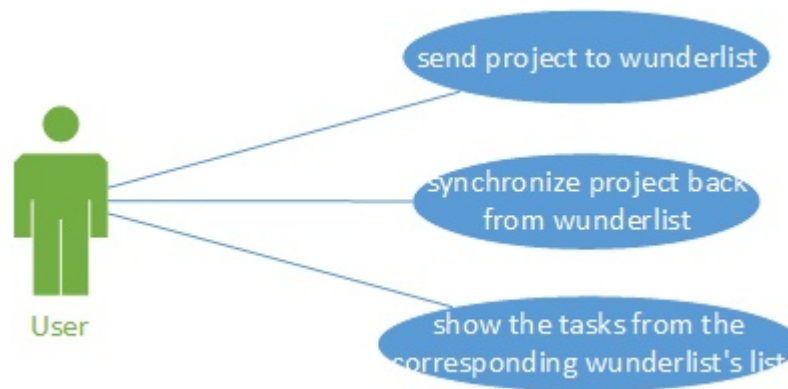


Figure 4: Use Case Chart

Analyzing the Case Model above, the functional and non-functional requirements that need to be satisfied by the Add-in are as follows:

#### Managing Users:

##### Functionl Requirements:

[FR 1.1] Enter the access credentials to Wunderlist.

##### Non-functionl Requirements:

[NFR 1.1] Check the user's license and act upon its type.

[NFR 1.2] Ensure security of the saved user's access credentials for Wunderlist.

**Managing Synchronization:****Functionl Requirements:**

[FR 2.1] Send a project to a list in Wunderlist and map each task to a task there.

[FR 2.2] Share the project on Wunderlist with the same resources in Wunderlist such as on MS Project.

[FR 2.3] Export the physical completion of the project tasks from Wunderlist and update them in MS Project.

[FR 2.4] Allow the user to view, through a third button, to view the exact information in Wunderlist without synchronizing them yet.

**Non-functionl Requirements:**

[NFR 2.1] Save the mapping table of the corresponding GUIDs of lists and tasks in MS Project and Wunderlist in lists in Project.

[NFR 2.2] Keep the mapping data permanent in MS Project to protect it from loss in case of re-installation.

[NFR 2.3] Do not affect the MS Project usability by rendering the lists hidden from the user.

## 3.2 Approach

### 3.2.1 Concept

The main idea of the solution is to map a project on MS project to a list Wunderlist. A list in Wunderlist is a collection of tasks, to which the the mapped project's tasks is also going to be mapped. Users interacting with Wunderlist can furthermore subdivide tasks to subtasks, assign tasks to different people, or also share the list with other people as needed, who might as well not be registered in MS Project itself. These changes, though, will not have effects on how the project is set-up in MS Project; it is considered the main data source of the shared projects. However,

the only aspect manner according to which the add-in can change in MS Project's projects according to the changes in their corresponding lists in Wunderlist is the physical completion of tasks. It will get updated according to how it is progressing in Wunderlist.

In other words, the add-in will share the project information with a Wunderlist's list. Subsequently, users will interact with the list in Wunderlist. And finally when the add-in is requested to update the project's data on MS Project, it will export the tasks' physical completion from Wunderlist and insert it into MS Project.

### **3.2.2 User interface**

According to the Product Functions analysis previously taken place, the user interacts with the Add-in in two different contexts as follows:

**Add-in Registration to Wunderlist** The first of which is when the user first installs the Add-in, in which the add-in should guide the user through the steps of registering the Add-in in Wunderlist. Upon its successful completion, the user can proceed to the next user interface scenario. Since, by default once the Add-in is installed to the user's MS Project account. It automatically redirects the user to the default.aspx webpage previously shown in the breakdown of the Add-in schema. It is quite a straight-forward decision to place the registration and authentication process guide of Wunderlist at default.aspx.

**Project Synchronization Actions:** The second interaction the user has with the Add-in would be from the MS project's project page, at which the user is to decide whether to share the project to Wunderlist or synchronize the information back from Wunderlist or just view a table of the corresponding list of tasks in Wunderlist. These functions boil down to having an interface of three buttons one for each function just described, in addition to a table showing the tasks and their corresponding attributes that are helpful for the user. A status message is also decided to be added to the interface to notify the user of all the possible progress state of the Add-in during the synchronization process.

## **3.3 Functional Requirements**

### **3.3.1 Scenarios:**

Here are the scenarios of a user who starts to use the add-in for the first time:

#### **3.2.1.1. User sends a project's information to Wunderlist**



User sends a project's information to Wunderlist	
Code	SCS001
Description	Describing how the user sends project's information to Wunderlist
Assumptions	1. User has access to a project on MS.
<p>Marco manages one of the company's projects on MS Project. He wants to send the project tasks to Wunderlist so that it is easily managed by users working on it, who some of do not even have access to MS Project. He clicks on the Send Button in the ribbon bar added by the add-in AgileBridge. AgileBridge sends the tasks and subtasks, which are not summaries, with their breakdown and deadline dates to the Wunderlist administrator account. Moreover, it shares it with the account of the Marco and all those the project is shared with on MS Project. Marco, then, receives a notification from his Wunderlist account to accept the sharing of the list, which has the same name as the project on MS, to his account. He accepts the invitation and shares the list with all employees working on this project that do not have access to it on project.</p>	

### 3.2.1.2. User synchronizes a project's information from the corresponding list from Wunderlist

User synchronizes a project's information from the corresponding list from Wunderlist	
Code	SCS002
Description	Describing how the user can synchronize back project's information from Wunderlist to MS Project
Assumptions	1. User has access to a project on MS. 2. The project has been already been shared as a list on Wunderlist.
<p>Since that last time Marco sent the project as a list on Wunderlist. Many workers have completed some of the tasks sent to them on Wunderlist. He, then, wants to update the data on MS Project according to the latest updates accomplished by them. Marco clicks on the second button added at the ribbon bar by the add-in called "From Wunderlist". The button updates the information on MS Project according to the data of Wunderlist and pops-up a modal showing all the tasks in the corresponding lists in Wunderlist. This table of tasks displayed in the modal also includes all the subordinate tasks and subtask only created on Wunderlist that do not have corresponding counterparts on MS Project to give him a more detailed view on the progress.</p>	

### 3.2.1.3. User makes changes to the project's planning on MS Project and resends it

User makes changes to the project's planning on MS Project and resends it	
Code	SCS003
Description	Describing how the user updates information on MS Project and send updates to Wunderlist
Assumptions	1. User has access to a project on MS. 2. The project has been already been shared as a list on Wunderlist.
<p>Marco has changed the planned schedule of the project. Therefore, some tasks deadlines are going to be delayed. Some tasks are deleted, some new ones are added. Marco, first, clicks the second "To Wunderlist" button so as to update the project with all the completion updates from Wunderlist. Then, he does the changes on the project and clicks the first button "To Wunderlist" to send the changes in the plans to Wunderlist. Wunderlist now has all the updated deadlines and breakdown of tasks according to the new order.</p>	

#### 3.2.1.4. User makes changes to the project's physical completion on MS Project and resends it

User makes changes to the project's physical completion on MS Project and resends it	
Code	SCS004
Description	Describing how the user updates information on MS Project and send updates to Wunderlist
Assumptions	1. User has access to a project on MS. 2. The project has been already been shared as a list on Wunderlist.
<p>This time, for some reason or another, Marco has exceptional changes in the completion of the project's tasks and wants to enter them in MS Project directly. Marco, first, clicks the third button "Task Table" button. The table pops-up a modal that shows the information, deadlines, break-down, and completion rates of all the tasks in Wunderlist. Marco studies the conflicts between the current state of the project in Wunderlist and the changes he wants to impose. He, then, resolves the conflicts either by first clicking the second button and then putting his modifications or directly from the beginning on the MS Project's project page. Finally, he clicks the first button "To Wunderlist" to send all the modification to Wunderlist as to allow all users to see the new synchronized version.</p>	

#### 3.2.1.5. User views the task table of the corresponding list in Wunderlist

User views the task table of the corresponding Wunderlist in Wunderlist	
Code	SCS005
Description	Describing how the user can view all the task of the corresponding Wunderlist in MS Project
Assumptions	1. User has access to a project on MS. 2. The project has been already been shared as a list on Wunderlist.
Marco would like to see all the updates done to the project in Wunderlist and all the subordinate tasks and subtasks that employees added in Wunderlist and their completion status and deadlines. To accomplish this, he clicks on the third button called "Task Table", which pops-up a modal showing all these data from Wunderlist.	

### 3.3.2 Flow Chart Models

#### 3.2.2.1 Authentication:

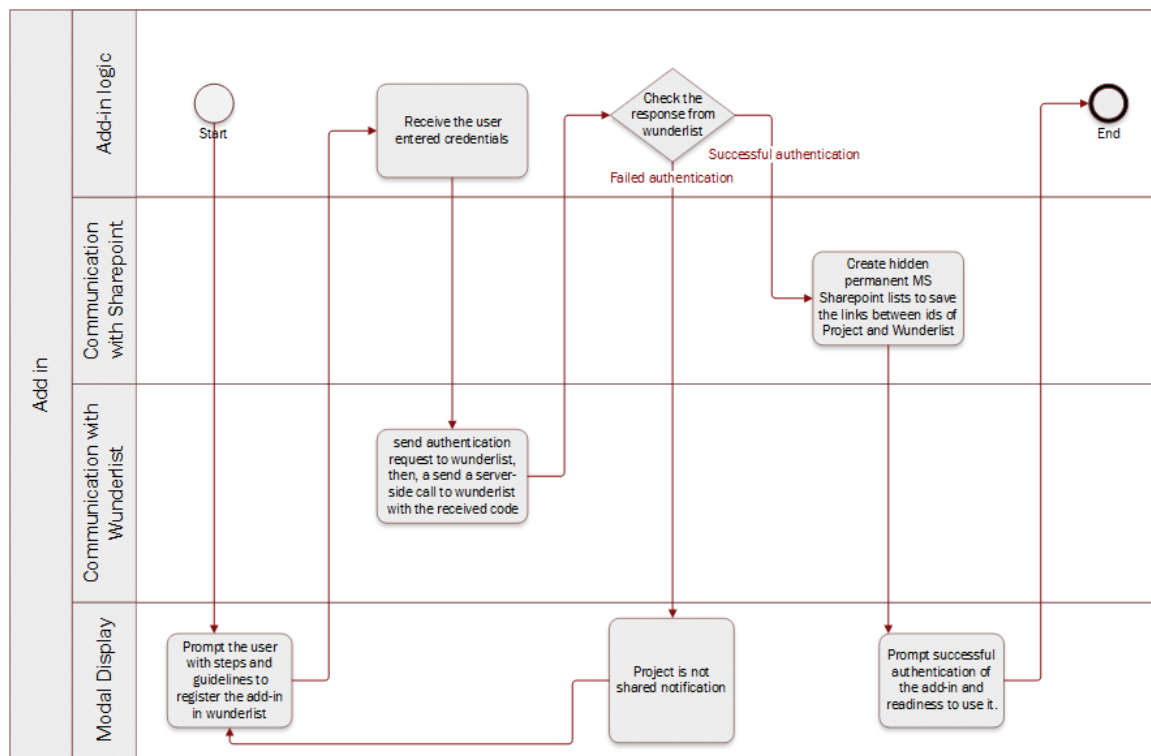


Figure 5: Authentication Flowchart

The figure above explains the flow of the authentication logic. Once the user installs the Add-in, the default page of the add-in will display the guidelines of how to register the add-in to the Administrator Wunderlist account providing the URLs

to be entered at the Wunderlist Registration page and asking the user to fill in back the credentials given by Wunderlist. Once the user hits save, the Add-in will send calls to Wunderlist to authenticate the Add-in. If the authentication fails, the guidelines page is redisplayed for the user; however, upon successful authentication with Wunderlist, the default page proceeds to create two hidden permanent lists in MS Projects that will hold the mapping between MS projects and Wunderlist's lists, and the other stores the mapping between MS tasks and Wunderlist tasks by saving their Ids. Once all of this is terminated successfully, the user is prompted by this success and that he/she can start using the add-in buttons now.

### 3.2.2.2 First button:

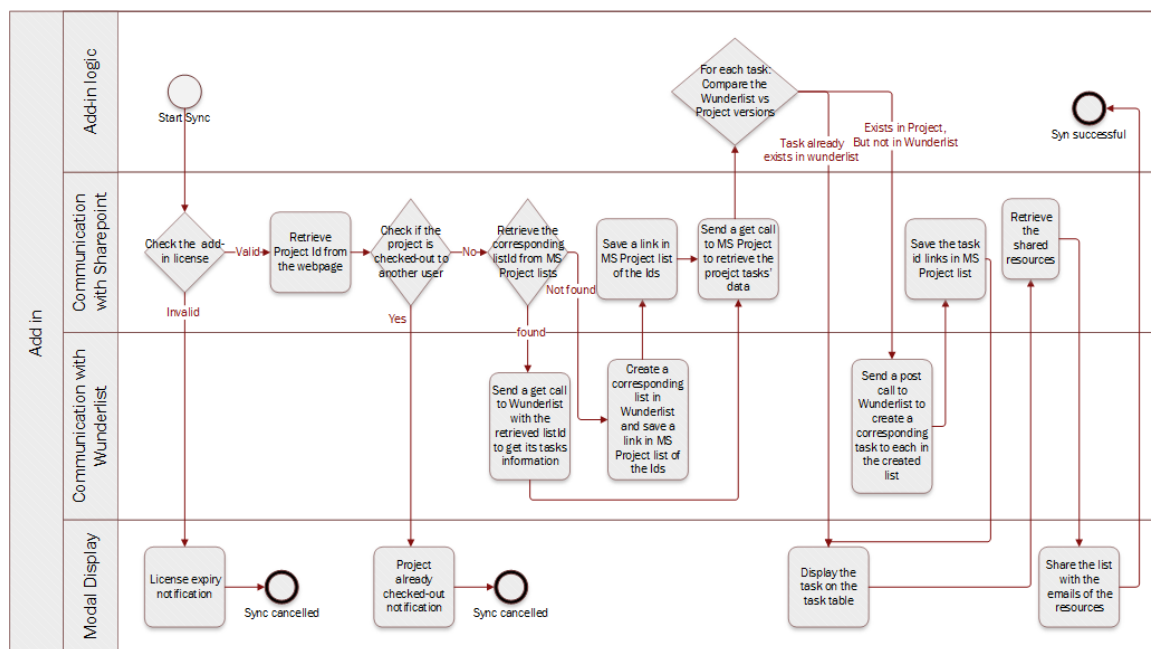


Figure 6: The first button's flowchart

The figure above provides the details flow chart of the actions taken once the first button is clicked. First, the license is checked to prove its validity. Then, it retrieves the current project's Id and checks if it had already been synchronized with wunderlist or not.

If it has not been synchnoized, a list is made with the same title of the project in Wunderlist and then all the tasks are mapped to tasks in that list and the project is shared with all its members of MS Project.

If it was already synchronized, then the project tasks are retrieved from both MS

Project and Wunderlist's list. then, a comparison is taken place to see which tasks are new in MS Project that needs to be added to Wunerlist and which task details were changed in Wunderlist that needs to be reset to its original form like in MS Project. Then, the wunderlist's list is shared with any new member of the project in MS Project if any. Finally, it displays all the tasks found in the Wunderlist's list on the Modal's task table.

### 3.2.2.3 Second button:

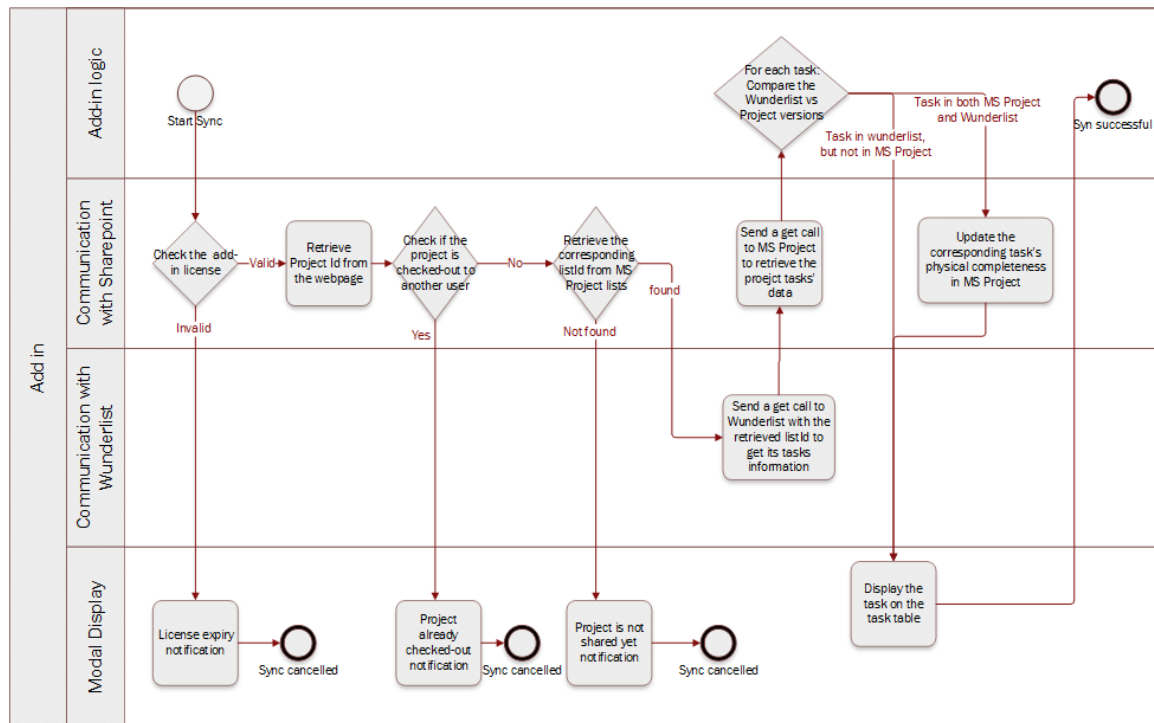


Figure 7: The second button's flowchart

Once the second button is clicked, the Add-in retrieves the users license. Once verified, the Add-in makes sure that the project is not checked-out to other users, and would notify the user if otherwise. Then, it proceeds to retrieve the Id of the Wunderlist's list Id. In case it was not found, it notifies the user of the need to synchronize it with Wunderlist first. In case a corresponding list is found however, the Add-in retrieves all the already mapped tasks from MS Project and Wunderlist and updates the physical completion of these tasks in MS Project according to their progress in Wunderlist. It also displays all the tasks found in the Wunderlist's list on the Modal's task table.

## 3.2.2.4 Third button:

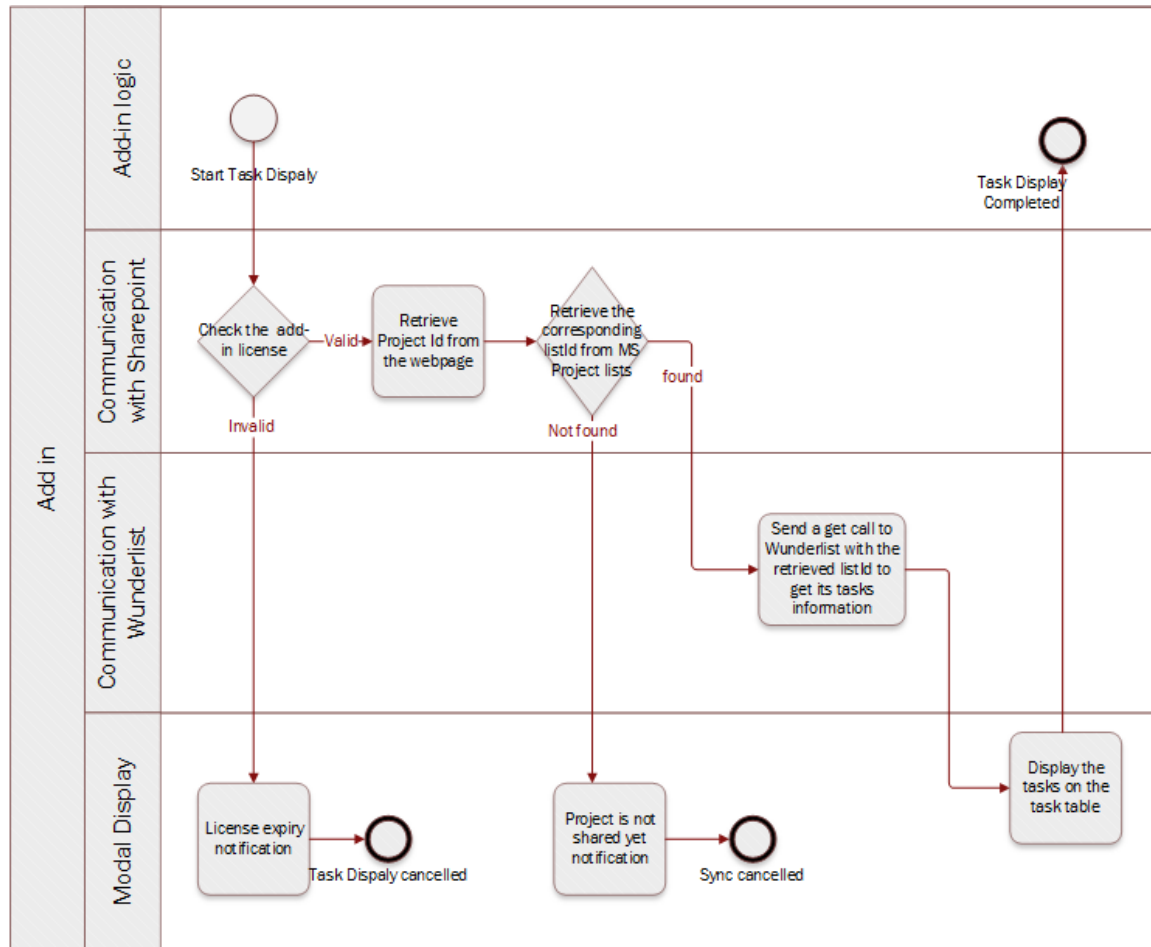


Figure 8: The third button's flowchart

Like the other two buttons, the first step is to check the Add-in license. Proven valid, the Add-in retrieves the current project's Id. Then retrieve the id of the corresponding Wunderlist's list. In case of being found, it calls Wunderlist retrieving all the details of the list's tasks and displaying them on the task table. In case, it was not found, a notification message stating that the current project has not been synchronized with Wunderlist yet or that the corresponding wunderlist's list was deleted. Hence, the user is in need to re-click the first button to synchronize it to Wunderlist.

## 4 Implementation

This section presents the implementation phase of the solution presented before according to the its theory provided in the previous section. First, the section enlists the libraries used, the Add-in development tool, its graphical interface. Then, it spots the light on the fundamental code snippets, which together puts the whole solution to work. Then, the section enumerates the main test cases applied to ensure correct functionality of the Add-in. The section, then, concludes with examples of the limitations and difficulties faced during the implementation phase.

### 4.1 Libraries used

jQuery: an open-source cross-platform JavaScript library designed to simplify the client-side scripting of HTML.

DataTables: a plug-in for the jQuery JavaScript library (<https://datatables.net>)[6].

### 4.2 Add-in Development

This section describes the tool used tool used to develop the Add-in, the Add-in structure, and the added components. It also presents some screenshots of the user interface.

#### 4.2.1 Add-in development Tool

The Sharepoint Add-in is developed as a project using the Microsoft developer tool "Visual Studio" [2]. The developed project is chosen in visual studio as a Visual C# =>Office/Sharepoint =>Add-ins =>Add-in for the Sharepoint.

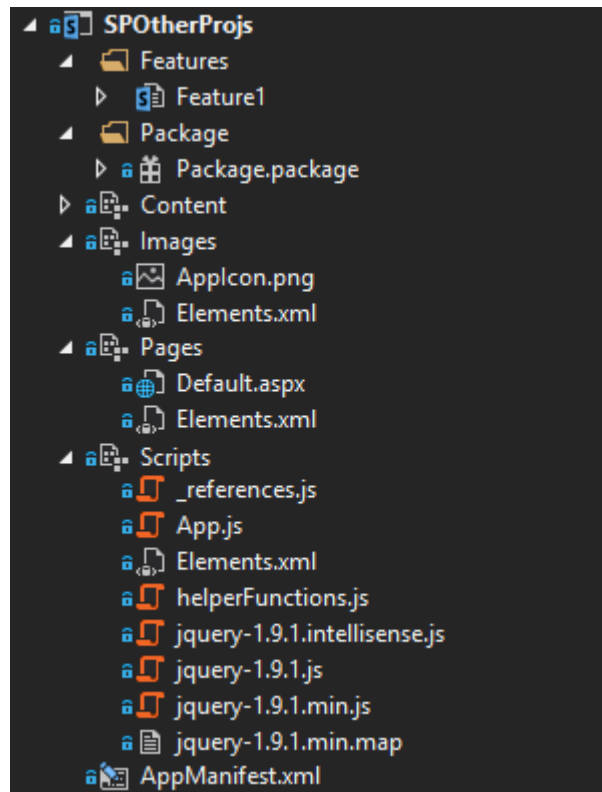


Figure 9: Add-in Code Structure

As shown in Figure 9, the structure of the add-in mainly relies on webpages accessible by the add-in, the default page already created by default. In addition, a scripts folder holds all the scripts the pages need to have access to. Also, the content folder and Images folder could hold referenced images and libraries, as desired.

The following step is to register the App to the user's Wunderlist account as seen in Figure 10. After which, the Add-in could now exploit Wunderlist's APIs to interact with Wunderlist sending and receiving data to the account it is registered to.



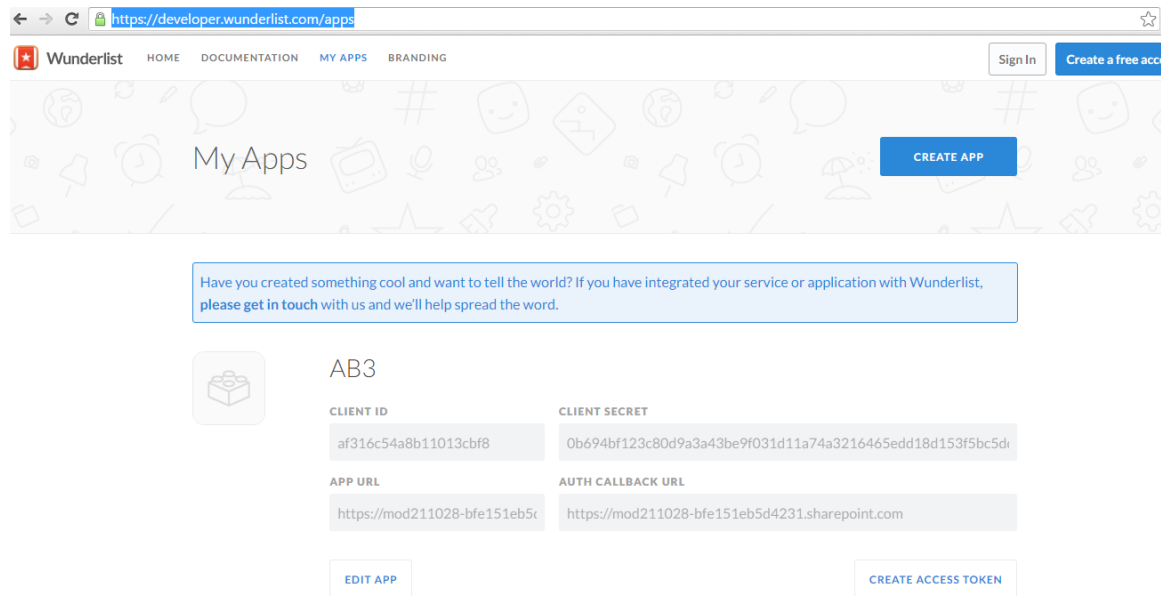


Figure 10: External App registration in Wunderlist

### 4.2.2 Add-in added items

According to the analysis done in the theory part. The add-in code structure shown in Figure 9 needed other items to be added to it. First A custom Ribbon Action item was added. This item includes XML file in which `commandUIExtensions` and their definitions could be wrote to define the ribbon group including the three buttons and a help button redirecting to the Add-in manual.

Another item to be added is a list in which the App would store the Client Id and Access Token used to communicate with the linked Wunderlist's accounts. These needed to be stored because the Add-in registration and authentication in Wunderlist is done once and for all; then, the credentials are to be stored in this list as to be used for all the upcoming communications.

Moreover, two permanent hidden lists are added to MS Project itself, not to the Add-in itself. These two lists are to contain the GUIDs of the MS Projects project and Wunderlist's list, and MS Project tasks and Wunderlist's tasks. The reason these lists are added directly to MS Project rather than the Add-in itself is that it is required that these mappings stay saved in Sharepoint itself so as not to lose them if the Add-in is uninstalled and re-installed again. The reason for this is that the Add-in's lists get erased once the Add-in is uninstalled from the tenant. The reason behind rendering the two lists hidden so as to keep them hidden from the user, this

way it does not confuse the user and at the same time they stay protected from the user's interference.

### 4.2.3 User interface

This section shows the development process of the interface of the Add-in.

**Add-in Registration to Wunderlist** Figure 11 shows the registration page that guides the user through the steps of registering the Add-in in Wunderlist. At the same page, the user subsequently, enters the credentials of this authentication to the Add-in to start the authentication process.

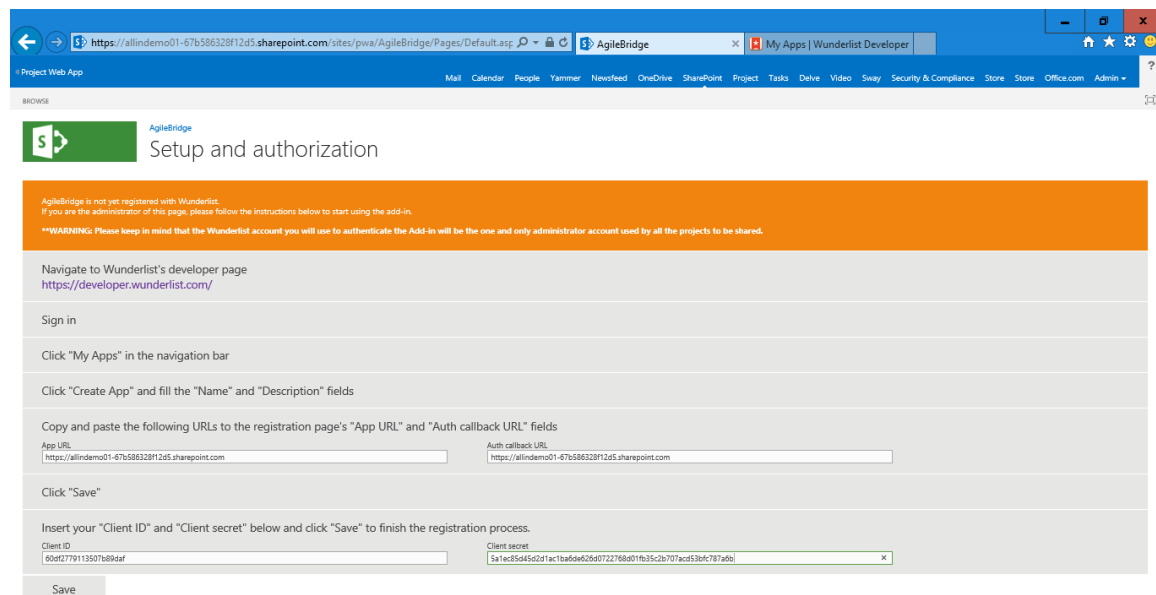


Figure 11: Process of Add-in Registration in Wunderlist

**Project Synchronization Actions:** In order to implement the three buttons explained in the Solution Theory section, the different typologies of Sharepoint Add-ins[3] were examined to spot out the suitable ones. There exist two possibilities:

1- The first is to make a web-part which can be added to the Add-in breakdown schema above as a new item =>Web Part[4]. Then, the project page is to be chosen as the location of the Web Part in the addition wizard. This finally displays the added page at the original project page in the figure below in between the Status message, which is under the ribbon bar, and the table of tasks and their timeline. Conceptually, the Web Part should include the three visual elements described before: The four buttons, a status message, and the table of tasks.

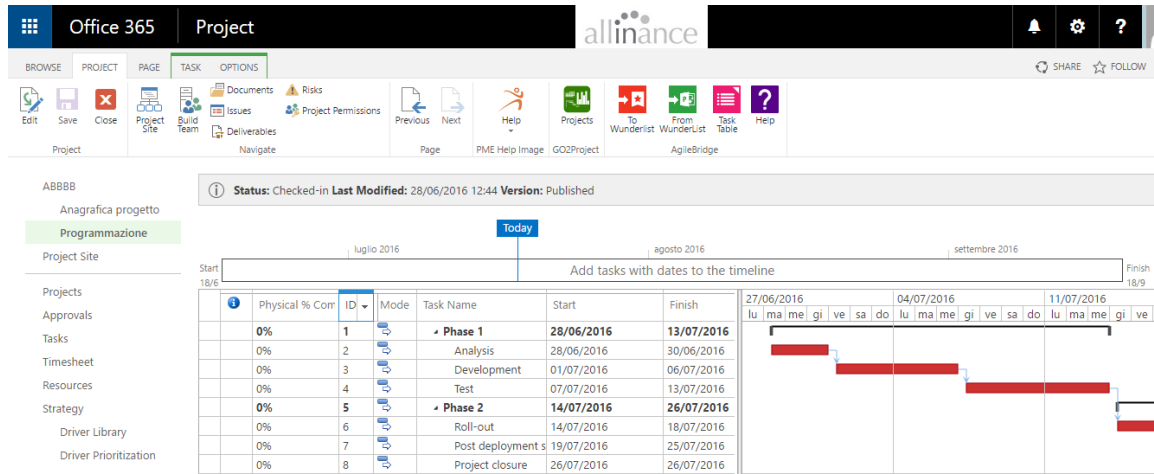


Figure 12: Project page of MS Project

2- The second possibility emerges from the privilege that the Sharepoint Add-in can add buttons to the ribbon bar, which can redirect to specific webpages that could be possibly added to the Add-in itself under the "pages" folder[5]. More research has shown that the ribbon also allows to pop-up webpages in the forms of "Modals". Hence, the concept would be adding four buttons to the ribbon bar using (Add new item =>Ribbon Custom Action). In the XML description added to describe these buttons, the position in the MS Project XML Schema could be specified. Their called functions could be added, which would redirect in the form of modals to webpages added to the Add-in. This means the same project page will stay as the background; however, the redirected-to webpage would pop-up in the form modal such as show in the figure below.



Figure 13: MS Project modal

In the beginning, the second user interface possibility was chosen to be developed; however, the project set off implementing the first option until adequate knowledge on how to implement the second option was collected, and then finally, transformed to its implementation.

The following three figures are screenshot of what the final implementation of the three buttons look like.

#### **Clicking the first button:**

This action will share the project to a sharepoint list, and each non-summary task in MS Project to a task in the wunderlist's list. The information shared would be its title, its WorkBreakDownStructure, and its due date. The wunderlist's shared list would also be shared with all the resources of the MS Project's shared project.

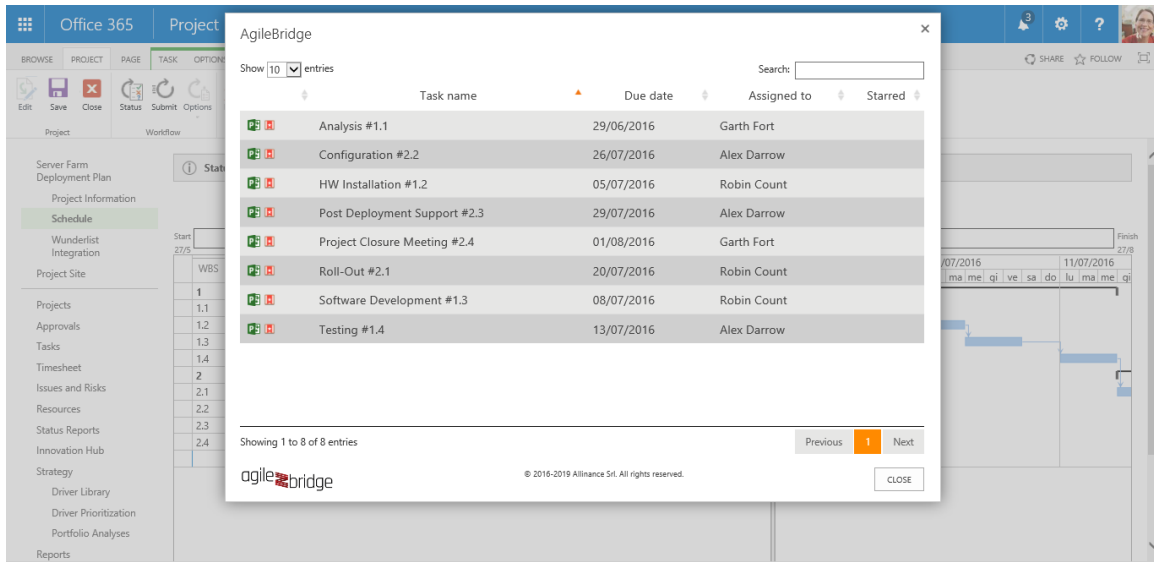


Figure 14: Modal shown for the First Button

**Clicking the second button:**

This action will fetch the tasks' data from Wunderlist and update the physical completion of the corresponding tasks in MS Project according to those in Wunderlist.

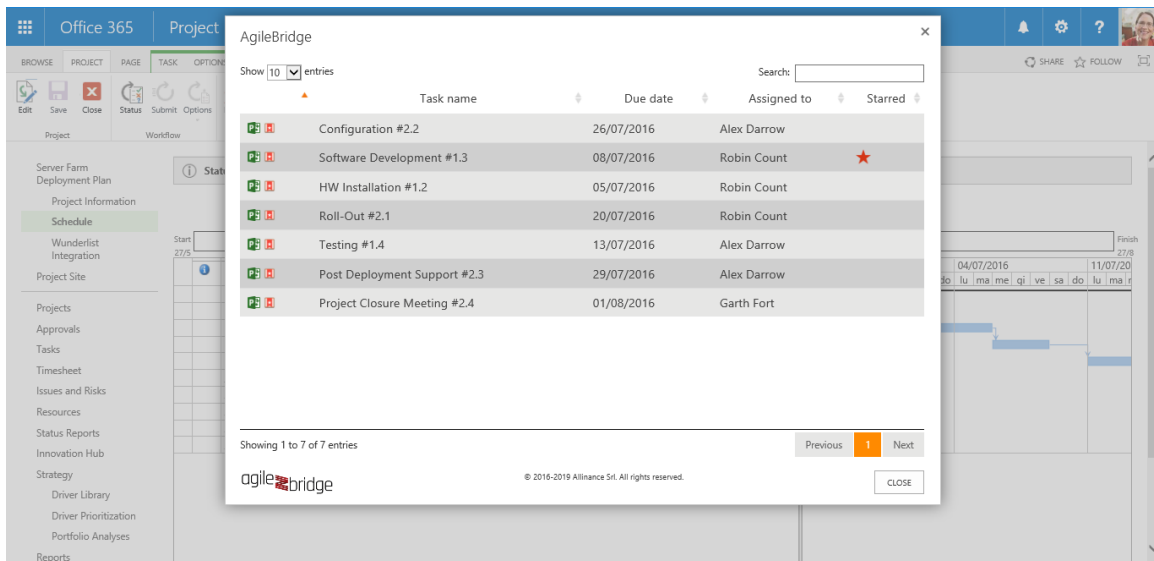


Figure 15: Modal shown for the Second Button

**Clicking the third button:**

This action will show a table of all the tasks in the corresponding list of the current MS Project's project in Wunderlist, including the ones that were added at the Wunderlist side with no counterpart in the current project itself.

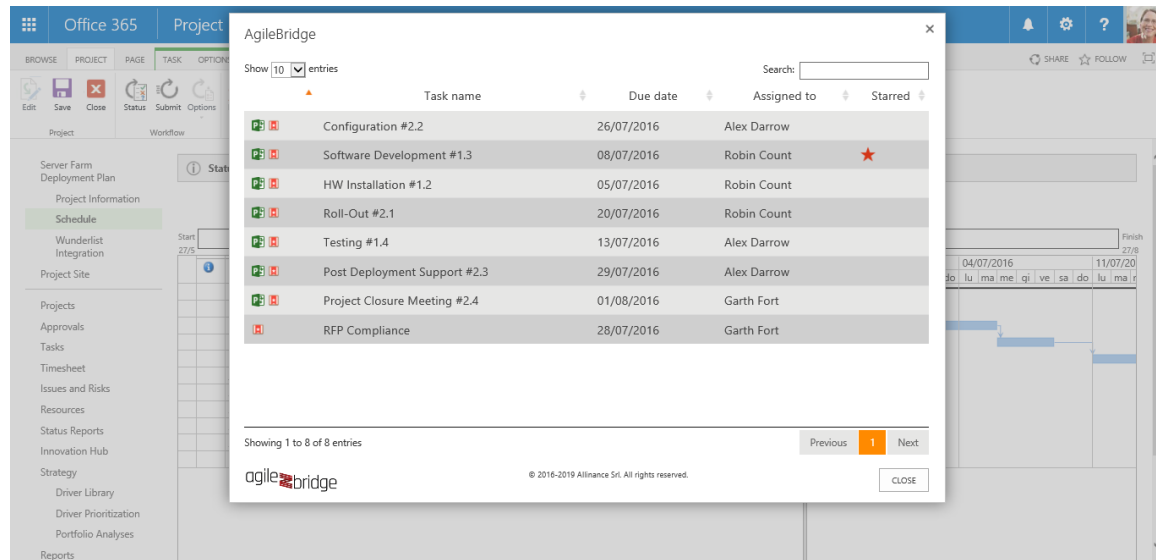


Figure 16: Modal shown for the Third Button

**The corresponding list in Wunderlist:**

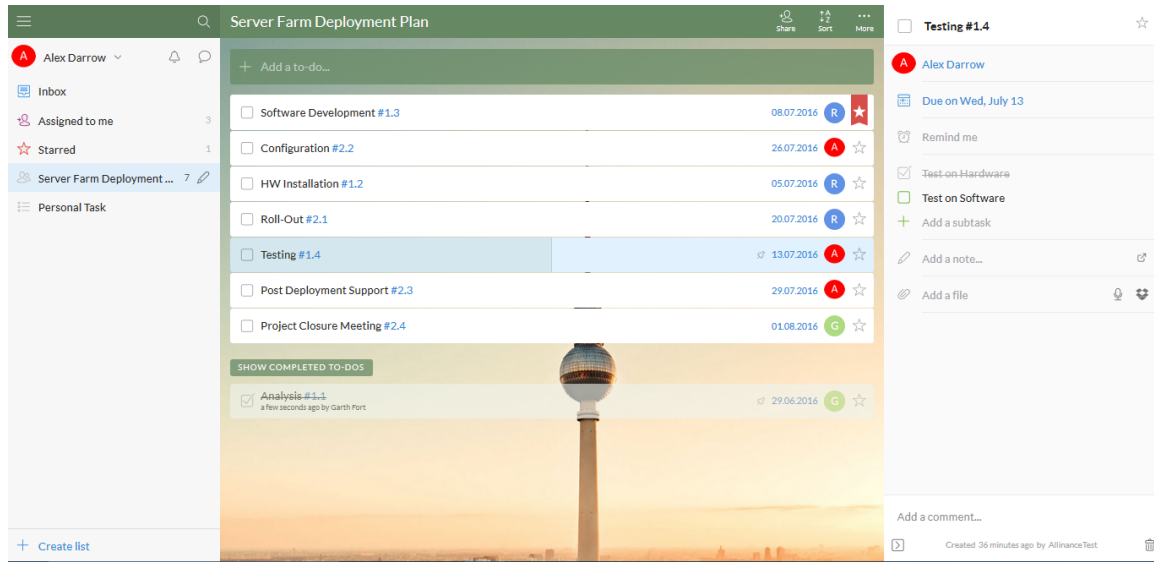


Figure 17: The project’s corresponding list in Wunderlist

## 4.3 Communication with Wunderlist

Wunderlist allows external application to interact with Wunderlist accounts without the need to handle the user’s password by sending calls containing X-Access-Token: OAUTH-TOKEN X-Client-ID: CLIENT-ID in the headers[7]. For the add-in to be able to communicate with Wunderlist, a two-step process has to be completed. It, firstly, needed to be registered in the user’s Wunderlist account to obtain the Client ID. Secondly, an authentication step is performed to obtain the Access Token mentioned above.

### 4.3.1 Registration of the Sharepoint Add-in

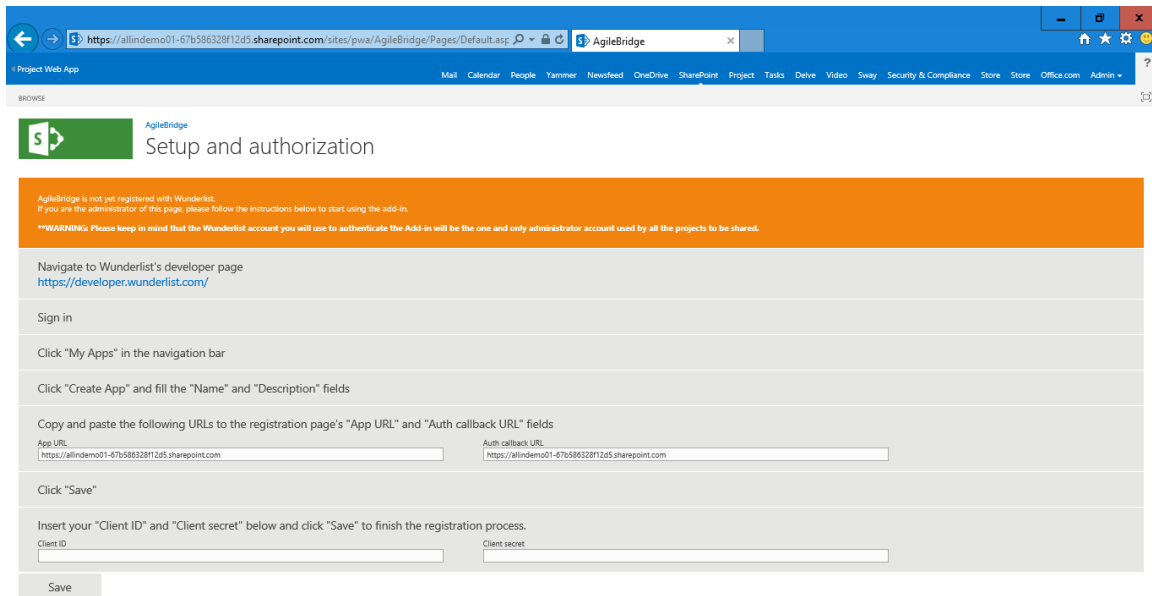
In order to be able to use the APIs of Wunderlist, the ”app” MS sharepoint add-in has to be registered under ”My Apps” in user’s Wunderlist account (<https://developer.Wunderlist.com/apps>)[7].

Due to security measures imposed by Wunderlist, this step has to be manually done by the user him/herself and research resulted in no-other through which this step could be automated with no need of the user intervention.

Therefore, the default page of the add-in once first added to the user’s Sharepoint will prompt a step-bystep guide/form on how to register and authenticate the add-in in their Wunderlist account. It is specifically required to copy the add-in website, which

is the URL to the add-in hosted on the sharepoint; for example (<https://allindemo01-67b586328f12d0.sharepoint.com>) in both the APP URL and the AUTH CALLBACK URL of the Wunderlist website previously mentioned.

Upon this registration, Wunderlist provides back the unique CLIENT ID and CLIENT SECRET, which the user needs to enter in the form at the default page of the add-in.



The screenshot shows a browser window with the URL <https://allindemo01-67b586328f12d0.sharepoint.com/sites/pwa/AgileBridge/Pages/Default.asp>. The page title is "AgileBridge" and the content area is titled "Setup and authorization".

**AgileBridge is not yet registered with Wunderlist.**  
If you are the administrator of this page, please follow the instructions below to start using the add-in.  
**\*\*WARNING: Please keep in mind that the Wunderlist account you will use to authenticate the Add-in will be the one and only administrator account used by all the projects to be shared.**

Navigate to Wunderlist's developer page  
<https://developer.wunderlist.com/>

Sign in

Click "My Apps" in the navigation bar

Click "Create App" and fill the "Name" and "Description" fields

Copy and paste the following URLs to the registration page's "App URL" and "Auth callback URL" fields

App URL <input type="text" value="https://allindemo01-67b586328f12d0.sharepoint.com"/>	Auth callback URL <input type="text" value="https://allindemo01-67b586328f12d0.sharepoint.com"/>
---	---

Click "Save"

Insert your "Client ID" and "Client secret" below and click "Save" to finish the registration process.

Client ID <input type="text"/>	Client secret <input type="text"/>
-----------------------------------	---------------------------------------

Save

Figure 18: Add-in Registration



### 4.3.2 Authenticating the Sharepoint Add-in and obtaining credentials

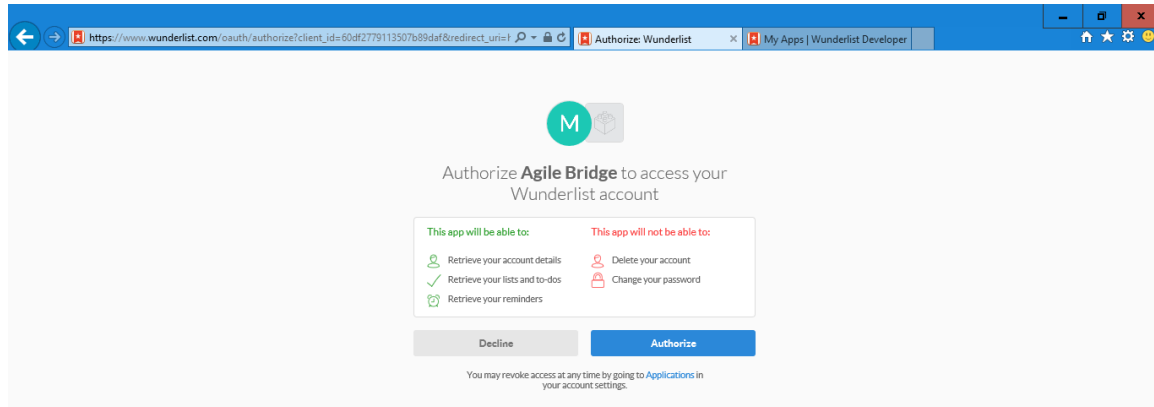


Figure 19: Add-in authorization Page at Wunderlist

Therefore, to complete, the authentication process: once the user clicks on the "SAVE" button, the page saves the user Wunderlist credentials in the Add-in List "API-DATA" and redirects to the Wunderlist's authorization URL:

$$\begin{aligned}
 & \text{"https : //www.Wunderlist.com/oauth/authorize?client_id = " + id +} \\
 & \text{"&redirect_uri = " + currentPage + "&state = " + appweburl"}
 \end{aligned}$$

Here, the state is supposed to be a random value, for which "appweburl" is used and the reason is that it is in fact made randomly by MS Project for every new app. This randomness protocol is used by Wunderlist to protect against cross-site request forgery attacks. When the authorization is done successfully at the part of Wunderlist, it redirects back to the same default page of the add-in adding a token called "Code" in the URL to be checked for the next call to Wunderlist. The next step is to send a server-side call to Wunderlist with the Client Id and Client Secret to get back the Access Token. The only obstacle is that it needs to be a server-side call directed to Wunderlist, but the add-in is loaded on the client's browser, which authentication calls are client-side not accepted by Wunderlist. Therefore, a proxy server site was built to act in the middle between the Add-in and Wunderlist. In the sense, that it receives a call from the Add-in and send the Access Token request call on behalf of the Add-in and respond to the Add-in with the received response from Wunderlist.

In more details, the add-in sends a POST call to the proxy website including the Client ID, Client Secret, and Code; which in turn is then redirected to Wunderlist by the proxy. At this point, Wunderlist verify the three credentials and respond with the Access Token to be stored by the Add-in.

In correspondence with the design decisions, this step is done once for all users by the administrator of the sharepoint, who is notified to use an administrator Wunderlist account, which will save much confusion if each user would connect his Microsoft account to different Wunderlist accounts.

### 4.3.3 Making calls to Wunderlist

Thanks to the registration and authentication steps above, now the add-in can successfully communicate with Wunderlist making REST calls identifying itself only by including the access token and client ID in the calls' headers.

### 4.3.4 GET calls:

The next two calls represent the actions through which the add-in sends information to Wunderlist using REST calls of type "GET" [8]

**Get a list from Wunderlist:** This call is used after the Add-in retrieves the corresponding list in Wunderlist to make sure it exists. The call is simply directed to URL: "https : //a.Wunderlist/api/v1/lists/" + the Id of the list in Wunderlist. If the Add-in gets a successful response, it means the lists exists; otherwise, a 404 response is received to indicate it does not exist, or that it was deleted from Wunderlist, in case it existed before.

**Get all tasks of a list from Wunderlist:** After making sure the lists exists in Wunderlist, a call is needed to get all the tasks and their details from this Wunderlist's list. This could be obtained through a call to URL: "https : //a.Wunderlist.com/api/v1/tasks?list\_id = " + the Id of the list in Wunderlist. The response is formatted in a JSON object and is received by the Add-in, upon successful processing by Wunderlist.

### 4.3.5 POST calls:

The next three calls represent the actions through which the add-in sends information to Wunderlist using REST calls of type "POST":

**Create a list in Wunderlist:** The Add-in needs to create a list in Wunderlist when the first button is clicked for a project for the first time, or whether for some reason the corresponding list in deleted and a re-send is needed. The Add-in then sends an Ajax call including the Access Toke and the Client ID which are stored by the program in the authentication internal list. The call is to be directed to the URL: "https : //a.Wunderlist.com/api/v1/lists". Upon successful sending of the call, Wunderlist responds with the Id of the list. The Add-in then goes to the

Project-list match permanently stored list in MS Project and adds/updates the entry corresponding to the current project.

**Create a task in Wunderlist:** When the first button is clicked and after creating the list in Wunderlist and receiving its ID, as explained in the first-button's flowchart Figure 6, the Add-in will loop for each non-summary task in the project and send all of them; in the case of re-sending the project, the Add-in will only send the new not-sent-before tasks. The call will include the list's Id in Wunderlist, the title of the task, its WorkBreakDownStructure in MS Project, and its due date. The call is directed to the URL: "*https://a.Wunderlist.com/api/v1/tasks*". Upon successful addition on Wunderlist, it responds to the call with the task's id in Wunderlist. This task's id is, then, stored in the task-matches list in MS Project alongside the task's id in MS Project and the corresponding Project's Id.

**Share a list with members in Wunderlist:** After the completion of the two types of calls above, the Add-in proceeds to sharing the list with all the (human)resources of the project. This is simply done by sending the Ajax call with two parameters: the list ID, and the email of the resource and repeating the call for all the required resources directed to the URL: "*https://a.Wunderlist.com/api/v1/memberships*". Wunderlist, then, shares the list with the account corresponding to such e-mail.

#### 4.3.6 PATCH calls:

**Update a task in a list in Wunderlist:** when the first button is re-clicked for the same project, then Add-in would retrieve the tasks information from both MS Project and Wunderlist and compares them. In the case, a task's data were changed in MS Project, the same changes must be mapped to the corresponding one in Wunderlist. This could be done by sending a call of type PATCH to Wunderlist at the URL: *https://a.Wunderlist.com/api/v1/tasks/* + task's Id. The body would then contain the revision of that task in Wunderlist, which would have already been received from Wunderlist, and the fields to be changed, be it the title, WorkBreakDownStructure, or due data or any mix of them.

## 4.4 Communication with Project online/server

For the Add-in to interact with MS Project, first, the pages displayed in the Modals have to include a series of script files loaded with MS Project itself[9]. The most important of these are the "SP.js" and "PS.js" files. The first one contain the request executor which is the class implements calls to MS Project lists. The second takes charge of asynchronous calls to get information about MS Project projects and user information.

#### 4.4.1 Retrieve project information:

**Project Context:** Having included the "PS.js" file in the scripts of a page, then the project context could be instantiated as an object from the class of the same name passing the URL of the tenant to its constructor. This Project Context could, later, be used to get information about all the projects in the tenant and their information.

#### 4.4.2 GET Calls

Having included the "SP.js", then all the next GET calls could be undertaken defining a Request Executor taking the Add-in Url as a parameter as follows *SP.RequestExecutor(appweblink)*. Then using this executor to run asynchronous calls.

**Retrieve an entry form an MS Project list:** To implement such a call, the body of the asynchronous call would be empty and all the passed information and requested data would be presented in the URL. The next call url represents a request for the entries with the field Project\_ID the same as the passed value from the list named "AgileBridge\_Config\_Project\_Wl\_links".

```
appweblink + /_api/SP.AppContextSite(@target)/web/lists/getbytitle
('AgileBridge_Config_Project_Wl_links')/items?@target = ' + hostUrl
+ '&$filter = Project_ID eq 'projectId'
```

#### 4.4.3 POST Calls:

**Build the hidden permanent lists:** To build the two lists which are used to save the connections between MS Project and Wunderlist, a client context is first initialized using the "Sp.js" classes. Then, from this a host context is initialized to be used to define list and its creation information such as: title, list type which is generic in our case, its hidden parameter, and all of its fields.

**Put an entry in a list:** Adding an entry to a list in MS Project is very similar to getting an item from a list; except that, the body would rather contain the data of each list field to be added and the url would have a query parameter to a specific field in the list.

```
appweblink + /_api/SP.AppContextSite(@target)/web/lists/getbytitle
('AgileBridge_Config_Project_Wl_links')/items?@target = ' + hostUrl
```

The data part of the call's body would look like:

```
"Title" : "Match", "Ptask_ID" : pTaskId, "Wltask_ID" : wlTaskId, "Project_ID" : projectId
```

**Delete an entry from a list:** Deleting an item from a list is a bit trickier than both querying or adding an item. To delete an item from a list, the call is of type 'POST'; however, the header will define an X-HTTP-Method of 'DELETE' and the title will have the id of the row itself in the list. Hence, first a query is done to the MS Project's list to get the row to be deleted. Then, the id is extracted from this data and included in the url of the delete call previously explained.

## 4.5 Non-functional requirement implementations

### 4.5.1 Check license

Checking the license is done by instantiating a `getlicenseInformation` object according to the client context and the GUID of the Add-in. This will return all the licenses the current user has for the Add-in. the latest one is, then, chosen, if any; then, the license still needs to be verified as authentic by MS Office's verification service according to the add-in license query and validation flow[10]. This helps avoid the case when Sharepoint might be tampered with. So a web request is sent to `<https://verification.service.officeapps.live.com/ova/verificationagent.svc >` to verify the license's authenticity.

For this, the website `<https://verification.service.officeapps.live.com/ova/verificationagent.svc >` is added as an endpoint in the `AppManifest.xml` file of the Add-in.

### 4.5.2 Minifying and obfuscating the JavaScript files

The fact that the Add-in is written in javascript and runs on web browsers means that any user can visualize all the files and make a copy of the code of the Add-in which will set the business value and licensing scheme useless. Therefore, the files need to be minified and obfuscated to at least make such savvy theft difficult to accomplish. This was done by first installing NodeJS Package Manager (NPM), Visual Studio Bundler and Minifier, and the Obfuscator packages. Then, make a routine such as shown in the appendix, which will firstly minify the JavaScript files and secondly obfuscate them. This routine was disabled at the time of debugging the code; but, later it was automatically re-ran before building the project according to a watch monitoring JavaScript file changes.

## 4.6 Testing

Here are the main test cases for which the Add-in was tested to assure proper functioning:

**4.6.1 Test Cases for the first button****Click the Send button for the first time**

Test Name	Click the Send button for the first time
Entry Condition	User at the project's page in MS Project and clicks the ribbon bar.
Expected Result	A list is created in the connected administrator account in Wunderlist, that would contain all the non-summary tasks with the correct title, WBDS, and due data. Moreover, the list is shared with all the accounts of the resources of the project.
Result	Verified.

**Add a task in MS Project and re-click the send button**

Test Name	Add a task in MS Project and re-click the send button
Entry Condition	User at the project's page in MS Project, having just added a new task to a project which was already sent to Wunderlist and clicks the ribbon bar.
Expected Result	The connected list had a new task added which maps all the details of the new task added in MS Project.
Result	Verified.

**Change a task's detail in MS Project and re-send it**

Test Name	Change a task's detail in MS Project and re-send it
Entry Condition	User at the project's page in MS Project, having just changed the details of a task of a project which was already sent to Wunderlist and clicks the ribbon bar.
Expected Result	The connected list have the details of the task, which links to the changed task, changed in line with the new details in MS Project.
Result	Verified.

**Change a task's detail in Wunderlist and re-send it**

Test Name	Change a task's detail in Wunderlist and re-send it
Entry Condition	User at the project's page in MS Project, having just changed the details of a task in Wunderlist of the list mapped to the same project and clicks the ribbon bar.
Expected Result	The connected list have the details of the changed task reverted back to the details representing it in MS Project.
Result	Verified.

#### **Add a new resource to the project and re-click the first button**

Test Name	Add a new resource to the project in MS Project and re-click the send button
Entry Condition	User at the project's page in MS Project, having just added a new resource to the project and clicks the ribbon bar.
Expected Result	The connected list gets shared with the Wunderlist's account of the new added resource.
Result	Verified.

#### **Delete a task in Wunderlist and re-click the send button**

Test Name	Delete a task in Wunderlist and re-click the send button
Entry Condition	User at the project's page in MS Project, having deleted a task from the Wunderlist's connected list and clicks the ribbon bar.
Expected Result	The connected list has the a new task with the same details as the old deleted task recreated all over again.
Result	Verified.

#### **Delete the corresponding list in Wunderlist and re-click the send button**

Test Name	Delete the corresponding list in Wunderlist and re-click the send button
Entry Condition	User at the project's page in MS Project, having deleted the connected list from Wunderlist and clicks the ribbon bar.
Expected Result	The user is notified by a message in the Modal stating that the connected list has been deleted and asked to re-click the first button again if he/she wants to resend it to Wunderlist.
Result	Verified.

**4.6.2 Test Cases for the second button****Check a task complete in Wunderlist and click the second button**

Test Name	Check a task complete in Wunderlist and click the second button
Entry Condition	User at the project's page in MS Project, having checked a task as complete in the connected list in Wunderlist and clicks the ribbon bar.
Expected Result	The physical completion of this task changes to 100% in MS Project or a correctly calculated partial completion of its parent task, in case it exists.
Result	Verified.

**Change the details of a task in Wunderlist and click the second button**

Test Name	Change the details of a task in Wunderlist and click the second button
Entry Condition	User at the project's page in MS Project, having changed the details of a shared task in the connected list in Wunderlist and clicks the ribbon bar.
Expected Result	The table in the Modal shows the details present in Wunderlist; however, the tasks details stay the same in MS Project.
Result	Verified.

**4.6.3 Test Cases for all three buttons****Add a new task in Wunderlist and click any of the three buttons**

Test Name	Add a new task in Wunderlist and click any of the three buttons
Entry Condition	User at the project's page in MS Project, having added a new task to the connected list in Wunderlist and clicks the ribbon bar.
Expected Result	The table shown in the modal shows all the tasks from the connected list including the added one regardless of where it exists in MS Project or not.
Result	Verified.

**Test Display table sorting for both title and due date with all three buttons**



Test Name	Test Display table sorting for both title and due date
Entry Condition	User at the project's page in MS Project and clicks the ribbon bar.
Expected Result	The table re-arranges all tasks correctly according to their title or according to their due date.
Result	Verified.

### Test display options of the modal for all three buttons

Test Name	Test display options of the modal for all three buttons
Entry Condition	User at the project's page in MS Project and clicks the ribbon bar.
Expected Result	The table closes correctly without reloading the page when the added close button is clicked. The search returns the correct filtered result when searched and the pagination is working correctly.
Result	Verified.

### Check user's license

Test Name	Check user's license
Entry Condition	User at the project's page in MS Project and clicks the ribbon bar.
Expected Result	In case the user has an expired or no license whatsoever the Modal notifies him/her of the need to buy a license or renew the license.
Result	Verified.
Notes	To test this feature, testing App licenses of different statuses such as: trial, paid with different expiry dates were imported to MS Sharepoint using a call to their utility APIs.

#### 4.6.4 Test Cases for the fourth "Guide" Button

##### User clicks the fourth "guide" button

Test Name	User clicks the fourth "guide" button
Entry Condition	User at the project's page in MS Project and clicks the ribbon bar.
Expected Result	A new window page opens to the webpage that includes the Add-in's guide
Result	Verified.

## 4.7 Limitations, Difficulties, and Work-arounds:

**Limitations:** The first main limitation imposed on developing Sharepoint hosted Add-ins is the fact that it has to be totally developed in javascript. This did not allow the use of C# classes of MS Project and imposed the need to search for the equivalent javascript calls to get the information from MS Project.

The second limitation is the that MS Projects does not allow the buttons added to the Ribbon bar by the Add-in to only redirect other webpages, when clicked, but not run javascript, nor even send custom parameters while calling any other webpage. This made the buttons disconnected from the rest of the Add-in's code. The connection between the buttons and the code is a key point in the development of the Add-in. Therefore, in the beginning, the Add-in was implemented as a webpart so that the buttons were implemented in a webpage shown in the page of the project itself, not on the ribbon bar. Then, the trick of adding the line *HostWebDialog = "TRUE"* made the redirected-to page appear as a modal while not leaving the main page. This solved the problem as now the modal displays a webpage that can run the Add-in javascript code and also access the project's information through its parent frame.

**Difficulties faced:** Not only are Ribbon Bar buttons quite limited in usage and capabilities, but also placing them in the correct position in the ribbon part and making them functional was quite tricky.

First, since the ribbon bar schema is not provided online, a complete one was to be retrieved from an on-premise deployment to find and write the correct exact the position where to add it in the button XML definitions files.

Second, while the XML definition file of the button groups allows the definitions of handlers- which are the webpages to redirect to, it does not allow to group command definitions. Therefore, they are disabled by nature. A trick to overcome this problem is to use the command of another active group in the same webpage, which will render the added group enabled.

Third, the XML files has some problem to find the added icons to should be displayed as the buttons, since most probably these definitions gets added to MS Project's main schema which disconnects it from the Add-in itself. The way to overcome it is to provide the icon in a base64 format instead of supplying the path of where to find it.

Some of the other difficulties had to deal with page designs that worked on Chrome, but did not work with other Internet browsers such as Safari that needed to be changed to work on all of them. Another dealt with the way user information(Project context and Client Context) are retrieved from MS Project in case it is cloud-based or on-premise. The call parameters had to be changed to a general form to be able to work on both environments. Another would be implementing a date sort in the imported table library to correctly sort due dates considering them as a date format

instead of text. In addition, some work-arounds were implemented for various objectives such as adding another close button to the Modal page to close it without reloading the whole page, given that the default close button reloads the page, which reduces usability for the user. Another one is the extensive use of promises due to the high number of calls done by the Add-in to avoid the highly nested Ajax calls all over the code.

## 5 Conclusions and future work

### 5.1 Conclusions

In conclusion, Project Portfolio Management is such a powerful centralized management technique that is used by many companies so as to monitor their progress, projects simultaneously. This allows companies to prioritize between their projects and stay focused on their strategic goals, which is essential to stay competitive in the market and guarantee the company's survival. Hence the need for Software to automatize this process and make it more efficient appears which is confirmed by the big number of software in the market that provides this service. Such tools are quite complex, they manage projects' tasks, reports, timesheets, financial audits, business processes, resources working on different tasks. And, more importantly, they provide reports and dashboards for managers to give summaries of the progress of all projects and help visualize the progress in them. However, by examining some of the most popular PPM software in the market, namely, CA PPM and Microsoft Project, one can see the possibilities for further integration and different direction of developments that these tools can leverage on. For example, these tools have very little representation on the handheld device scene, especially for employees who enter the bulk of the data on project tasks completion. In the case of CA PPM, of all the very diverse functionalities they have they only provide apps to fill in personal timesheets and another to view reports. MS Project only provides apps for filling personal timesheets. However, Microsoft has already acquired Wunderlist, a task management tool with a very convenient, usable, user friendly mobile app. Hence, this works stems from this gap by integrating both MS Project and Wunderlist. This is done by developing a Sharepoint Add-in which sends information between Wunderlist and MS Project. This Add-in, therefore, fills the previously mentioned gap by leveraging the usability of Wunderlist's mobile app instead of developing an app from scratch just for MS Project. As a side result, users will even be able to better manage their tasks through Wunderlist with its simple interface more detailed functionalities for tasks compared to MS Project and then re-synchronize the progress with MS Project whenever wanted.

## 5.2 Future work

The work in this project is mainly about successfully building the core of the integration between MS Project and Wunderlist. However, further development could easily be added at the visualization level. In the sense that since all the information of all the mapped Wunderlist's tasks are already accessible by the Add-in, the displayed task table could present more details of the tasks besides their title and due date. For example, it would be handy for the user to be able the physical completion of each task as one of the columns on the displayed table.

So far, the methodology has been to show the most fundamental and required details of the mapped Wunderlist's tasks because showing a lot of details would not make much sense as it would be identical to Wunderlist. In such case, the client could just view the mapped list in Wunderlist itself rather than visualize it in MS Project. Therefore, adding more information could be done as required by the client; such as, displaying which task is a sub-task of which, if any on the table.

Another direction of more development to be considered is studying how to connect the project to more than one Wunderlist administrator account and resolving the clashes that could happen.

A third possible direction of development is the implementation of a different licensing typology that would consider controlling the number of and which resources to be allowed to use the Add-in.

## 6 Appendix

This section provides the main code snippets of the Implementation.

### 6.1 Coummunication with Wunderlist

#### 6.1.1 GET calls

Get a list from Wunderlist:

```
01. $.ajax(  
02.   {  
03.     url: 'https://a.wunderlist.com/api/v1/lists/' + wlId,  
04.     type: 'GET',  
05.     beforeSend: function (xhr) {  
06.       xhr.setRequestHeader("X-Access-Token", accessToken);  
07.       xhr.setRequestHeader("X-Client-ID", clientId);  
08.     },  
09.     success: function (response) {  
10.       existsCallback();  
11.     },  
12.     error: function (response) {  
13.       console.log(response);  
14.       if (response.status == "404")  
15.         deletedCallback();  
16.       else  
17.         $('#statusText').html('<h3>Failed List Retrieval.</h3>');  
18.     }  
19.   }  
20. });
```

Get all tasks of a list from Wunderlist:

```
01. $.ajax(  
02.   {  
03.     url: 'https://a.wunderlist.com/api/v1/tasks' + '?list_id=' + listId,  
04.     type: 'GET',  
05.     beforeSend: function (xhr) {  
06.       xhr.setRequestHeader("X-Access-Token", accessToken);  
07.       xhr.setRequestHeader("X-Client-ID", clientId);  
08.     },  
09.     success: function (response) {  
10.       var wlTasks = response;  
11.     },  
12.     error: errorHandler  
13.   }  
13. });
```

### 6.1.2 POST calls

Create a list in Wunderlist:

```
01. // Send to Wunderlist and get Wunderlist ID
02. $.ajax({
03.   url: 'https://a.wunderlist.com/api/v1/lists',
04.   type: 'POST',
05.   contentType: 'application/json',
06.   dataType: 'json',
07.   data: JSON.stringify({ "title": response.d.Name }),
08.   beforeSend: function (xhr) {
09.     xhr.setRequestHeader("X-Access-Token", accessToken);
10.     xhr.setRequestHeader("X-Client-ID", clientId);
11.   },
12.   success: function (response) {
13.     // Add ID pairs to the list and get tasks
14.     listId = response.id;
15.     $.getScript(scriptbase + "SP.RequestExecutor.js", function () {
16.       addLink(projectUid, listId);
17.     });
18.   },
19.   error: function (error) {
20.     console.log(error);
21.     clearMessage();
22.     console.log('Error creating list.');
```

Create a task in Wunderlist:

```

01. $.ajax({
02.     url: 'https://a.wunderlist.com/api/v1/tasks',
03.     type: 'POST',
04.     contentType: 'application/json',
05.     dataType: 'json',
06.     data: JSON.stringify(
07.         {
08.             "list_id": listId,
09.             "title": tasks_entry.Name + " " + "#" + tasks_entry.WorkBreakdownStructure,
10.             "due_date": tasks_entry.Finish
11.         }
12.     ),
13.     beforeSend: function (xhr) {
14.         xhr.setRequestHeader("X-Access-Token", accessToken);
15.         xhr.setRequestHeader("X-Client-ID", clientId);
16.     },
17.     success: function (response) {
18.         // Add Task ID pairs to the task_pair_list and get tasks
19.         var taskId = response.id;
20.
21.         $.getScript(scriptbase + "SP.RequestExecutor.js", function () {
22.             addTaskLink(ptaskId, taskId, projectId);
23.         });
24.     },
25.     error: function (error) {
26.         console.log(error);
27.         console.log('Error creating list. ');
28.         clearMessage();
29.     }
30. });

```

### Share a list with members in Wunderlist:

```

01. // share the list first with the current user
02. $.ajax({
03.     url: 'https://a.wunderlist.com/api/v1/memberships',
04.     type: 'POST',
05.     contentType: 'application/json',
06.     dataType: 'json',
07.     data: JSON.stringify({ "list_id": listId, "email": user.get_email() }),
08.     beforeSend: function (xhr) {
09.         xhr.setRequestHeader("X-Access-Token", accessToken);
10.         xhr.setRequestHeader("X-Client-ID", clientId);
11.     },
12.     success: function (response) {
13.     },
14.     error: function (error) {
15.         console.log(error);
16.     }
17. });

```

### 6.1.3 PATCH calls

#### Update a task in a list in Wunderlist:



```

01. $.ajax({
02.   url: 'https://a.wunderlist.com/api/v1/tasks/' + w1Task.id,
03.   type: 'PATCH',
04.   contentType: 'application/json',
05.   dataType: 'json',
06.   data: JSON.stringify(
07.     {
08.       "revision": w1Task.revision,
09.       "title": pTask.Name + " " + "#" + pTask.WorkBreakdownStructure,
10.       "due_date": pTask.Finish
11.     }
12.   ),
13.   beforeSend: function (xhr) {
14.     xhr.setRequestHeader("X-Access-Token", accessToken);
15.     xhr.setRequestHeader("X-Client-ID", clientId);
16.   },
17.   success: function (response) {
18.   },
19.   error: function (error) {
20.     console.log(error);
21.     console.log('Error updating the changed task info.');
```

## 6.2 Communication with Project online/server

### 6.2.1 Retrieve project information:

Project Context:

```

01. var contextUrl = hostUrl.substr(hostUrl.indexOf("/", 8));
02. projContext = new PS.ProjectContext(contextUrl);
03. projects = projContext.get_projects();
04. projContext.load(projects, "Include( Name, CreatedDate, Id, IsCheckedOut )");
05. projContext.executeQueryAsync(sync, function () { console.log("Failed to load project data."); });
```

### 6.2.2 GET Calls

Retrieve an entry form an MS Project list:

```
01. executor.executeAsync(  
02. {  
03.     url:  
04.         appweburl +  
05.         "/_api/SP.AppContextSite(@target)/web/lists/getbytitle('AgileBridge_Config_Project_W1_links')  
06.         /items?@target='" + hostUrl + "'&filter=Project_ID eq " + "" + projectId + "",  
07.     method: "GET",  
08.     headers: { "Accept": "application/json; odata=verbose" },  
09.     success: function (data)  
10.     {  
11.         var response = JSON.parse(data.body);  
12.         var projectFoundinWL = response.d.results.length;  
13.         var listId;  
14.         console.log('search for listid successful');  
15.         if (projectFoundinWL) {  
16.             console.log('listid found');  
17.             listId = response.d.results[0].List_ID;  
18.             console.log('trovato! id->' + listId);  
19.             refreshTable(listId);  
20.         }  
21.         else {  
22.             console.log('prj not found :(');  
23.             $('#statusText').html('<h3>The project isn\'t synced yet. ]</h3>');  
24.         }  
25.     }  
26. })
```

### 6.2.3 POST Calls:

Build the hidden permanent lists:

```

01. // Create an announcement SharePoint list with the name that the user specifies.
02. var hostUrl = decodeURIComponent(getQueryStringParameter("SPHostUrl"));
03. var currentContext = new SP.ClientContext.get_current();
04. var hostContext = new SP.AppContextSite(currentContext, hostUrl);
05. var hostWeb = hostContext.get_web();
06.
07. var listName = "AgileBridge_Config_Project_Wl_links";
08. var listToFind1 = hostWeb.get_lists().getByTitle(listName);
09. currentContext.load(listToFind1);
10. currentContext.executeQueryAsync(
11. function() {
12.     console.log(listToFind1.get_title()+ " list was found.");
13. },
14.     function(sender, args){
15.
16.         var theListName = "AgileBridge_Config_Project_Wl_links";
17.         // this is to create the AgileBridge_Config_Project_Wl_links list.
18.         var strNewFields = ["<Field Name=\"Project_x005f_ID\" DisplayName=\"Project_ID\"
19. Type=\"Text\" Required=\"TRUE\" />", "<Field Name=\"List_x005f_ID\"
20. DisplayName=\"List_ID\" Type=\"Number\" Required=\"TRUE\" />"];
21.
22.         var listCreationInfo = new SP.ListCreationInformation();
23.         listCreationInfo.set_title(theListName);
24.         listCreationInfo.set_templateType(SP.ListTemplateType.genericList);
25.         var lists = hostWeb.get_lists();
26.         var newList = lists.add(listCreationInfo);
27.         newList.set_hidden('true');
28.         newList.set_onQuickLaunch('false');
29.         newList.update();
30.         currentContext.load(newList);
31.         for (var i = 0; i < strNewFields.length; i++)
32.             newList.get_fields().addFieldAsXml(strNewFields[i]);
33.
34.         currentContext.executeQueryAsync(onListCreationSuccess(), onListCreationFail);
35.     }
36. );

```

Put an entry in a list:

```

01. executor.executeAsync(
02. {
03.     url:
04.         appweburl +
05.         "/_api//SP.AppContextSite(@target)/web/lists/getbytitle('"
06.         + listType + "')/Items?@target='" + hostweburl + "'",
07.     method: "POST",
08.     body: JSON.stringify({ "__metadata": { "type": entityType },
09. "Title": "Match", "Ptask_ID": pTaskId, "Wltask_ID": wlTaskId, "Project_ID": projectId }),
10.     headers:
11.     {
12.         "accept": "application/json; odata=verbose",
13.         "content-type": "application/json;odata=verbose"
14.     },
15.     success: createdSuccessHandler,
16.     error: errorHandler
17. });

```

## Delete an entry from a list:

```

01.   executor.executeAsync(
02.   {
03.       url:
04.         appweburl +
05.         "/_api/SP.AppContextSite(@target)/web/lists/getbytitle('AgileBridge_Config_Task_Project_W1_Matches')
06.         /items(" + listitem_toDelete.Id + ")?@target=" + hostUrl + "",
07.       method: 'POST',
08.       headers:
09.       {
10.         "Accept": "application/json; odata=verbose",
11.         "X-HTTP-Method": "DELETE",
12.         "If-Match": "*"
13.       },
14.       success: function (result) {
15.         console.log("Deleted " + listitem_toDelete.Id);
16.       },
17.       error: function (e) {
18.         console.log("Failed to delete. Error: " + e);
19.         console.log(e);
20.       }
21.   });

```

## 6.3 Non-functional requirements implementations

### 6.3.1 Check license

```

01.   //Retrieve license from SharePoint; change this productId with the one for your app; you can get it from AppManifest.xml
02.   licenseCollection = SP.Utilities.Utility.getAppLicenseInformation(lic_clientContext, '{f887d8df-a679-42f1-9577-61d891db5ab7}');
03.   lic_clientContext.executeQueryAsync(function () { onRetrieveLicenseFromSPSuccess(callback); }, onRetrieveLicenseFromSPFailure);
04.
05.
06.
07.   //Retrieval call succeeded (doesn't mean there is a license, look at the contents to see if there is one)
08.   function onRetrieveLicenseFromSPSuccess(callback) {
09.     if (licenseCollection.get_count() > 0) {
10.       topLicense = licenseCollection.get_item(0).get_rawXMLLicenseToken();
11.
12.       //debug
13.       console.log("License retrieved from SharePoint: \n" + topLicense);
14.
15.       //encode license; required to call the verification service since it will be sent on the URL
16.       encodedTopLicense = encodeURIComponent(topLicense);
17.
18.       //debug, display the encoded license
19.       console.log("License ready to be verified:\n" + encodedTopLicense);
20.
21.       //Call verification service via the WebProxy; this ensures the validity of the license in case SharePoint was tampered with
22.       var lic_request = new SP.WebRequestInfo();
23.       //We use the REST flavor of the verification service
24.       lic_request.set_url("https://verification.service.officeapps.live.com/ova/verificationagent.svc/rest/verify?token=" + encodedTopLicense);
25.       lic_request.set_method("GET");
26.       lic_response = SP.WebProxy.invoke(lic_clientContext, lic_request);
27.       // Set the event handlers and invoke the request
28.       lic_clientContext.executeQueryAsync(function () { onVerificationCallSuccess(callback); }, onVerificationCallFailure);
29.     }
30.     else {
31.       console.log("The user doesn't have a license");
32.       $('#statusText').html('<h3 style="color: tomato">The user doesn't have a license. </h3>');
33.     }
34.   }
35. }

```



## 6.3.2 Minifying and obfuscating the JavaScript files

```
01.     uglify: {
02.         options: {
03.             globals: {
04.                 jQuery: true
05.             }
06.         },
07.         scripts: {
08.             files: [
09.                 {
10.                     expand: true,
11.                     cwd: 'Scripts',
12.                     src: '*.js',
13.                     dest: 'Scripts/S',
14.                     ext: '.js',
15.                 }
16.             ]
17.         }
18.     },
19.     jsObfuscate: {
20.         options: {
21.             concurrency: 2,
22.             keepLinefeeds: false,
23.             keepIndentations: false,
24.             encodeStrings: true,
25.             encodeNumbers: true,
26.             moveStrings: true,
27.             replaceNames: true,
28.             variableExclusions: [ '^_get_', '^_set_', '^_mtd_' ]
29.         },
30.
31.         scripts: {
32.             files: [
33.                 {
34.                     expand: true,
35.                     cwd: 'Scripts/S',
36.                     src: '*.js',
37.                     dest: 'Scripts/O',
38.                     ext: '.js',
39.                 }
40.             ]
41.         }
42.     }
43. });
```

## References

- [1] *CA Clarity Project & Portfolio Manager: Administration Guide*, v12.1.0 ed., CA Technologies, 2015, [https://supportcontent.ca.com/cadocs/0/CA Clarity PPM 12 1 0-ENU/Bookshelf\\_Files/PDF/CAClarityPPM\\_AdministrationGuide\\_ENU.pdf](https://supportcontent.ca.com/cadocs/0/CA%20Clarity%20PPM%2012%201%200-ENU/Bookshelf_Files/PDF/CAClarityPPM_AdministrationGuide_ENU.pdf). 4
- [2] *Walkthrough: Create a SharePoint-hosted Project Server add-in*, Microsoft, dec 2015, <https://msdn.microsoft.com/en-us/library/office/jj873844.aspx>. 17
- [3] *SharePoint Add-ins*, Microsoft, may 2016, <https://msdn.microsoft.com/en-us/library/office/fp179930.aspx>. 20
- [4] *Walkthrough: Creating a Web Part for SharePoint*, Microsoft, sep 2015, <https://msdn.microsoft.com/en-us/library/ee231551.aspx>. 20
- [5] *Create custom actions to deploy with SharePoint Add-ins*, Microsoft, sep 2015, <https://msdn.microsoft.com/en-us/library/office/jj163954.aspx>. 21
- [6] *DataTables: Table plug-in for jQuery*, SpryMedia Ltd, <https://datatables.net>. 17
- [7] Wunderlist, *Wunderlist API Documentation(Authorization)*, 6Wunderkinder, 2015, <https://developer.wunderlist.com/documentation/concepts/authorization>. 25
- [8] *Wunderlist API Documentation*, 6Wunderkinder, 2015, <https://developer.wunderlist.com/documentation>. 28
- [9] *Complete basic operations using JavaScript library code in SharePoint 2013*, Microsoft, oct 2015, <https://msdn.microsoft.com/en-us/library/office/jj163201.aspx>. 29
- [10] *License your Office and SharePoint Add-ins*, Microsoft, nov 2015, <https://msdn.microsoft.com/en-us/library/office/jj163257.aspx>. 31